Efficient Cryptanalysis of Homophonic Substitution Ciphers

Amrapali Dhavare* Richard M. Low† Mark Stamp‡

Abstract

Substitution ciphers are among the earliest methods of encryption. Examples of classic substitution ciphers include the well-known simple substitution and the less well-known homophonic substitution. Simple substitution ciphers are indeed simple, both in terms of their use and their cryptanalysis. Homophonic substitutions—in which a plaintext symbol can map to more than one ciphertext symbol—are also easy to use, but far more challenging to break. Even with modern computing technology, homophonic substitutions can present a significant cryptanalytic challenge.

This paper focuses on the design and implementation of an efficient algorithm to break homophonic substitution ciphers. We employ a nested hill climb approach that generalizes the fastest known attack on simple substitution ciphers. We test our algorithm on a wide variety of homophonic substitutions and provide success rates as a function of both the ciphertext alphabet size and ciphertext length. Finally, we apply our technique to the "Zodiac 340" cipher, which is an unsolved message created by the infamous Zodiac killer.

Keywords: homophonic substitution cipher, simple substitution cipher, hill climb, heuristic search, Zodiac 340

1 Introduction

Substitution ciphers are among the oldest encryption methods—they are simple, intuitive, and widely studied [21]. Many variants of the substitution cipher have been developed, including monoalphabetic systems, which employ a fixed substitution, and polyalphabetic systems, where the substitution varies. Example of monoalphabetic substitutions include the simple substitution and homophonic substitution ciphers. Examples of polyalphabetic substitutions include the Vigenére cipher and

*Department of Computer Science, San Jose State University
†Department of Mathematics, San Jose State University
‡Department of Computer Science, San Jose State University: stamp@cs.sjsu.edu

World War II era rotor cipher machines, such as the Enigma. Here, we are concerned with monoalphabetic substitutions.

For the simple substitution cipher, the plaintext-to-ciphertext mapping is one-to-one. In contrast, for homophonic substitution ciphers, this mapping is one-to-many, that is, one plaintext symbol can be encrypted as multiple ciphertext symbols. For example, when encrypting, we could randomly choose from the allowed ciphertext symbols for each plaintext letter. Since each ciphertext symbol can map to only one plaintext symbol, decryption is straightforward and unique.

The simple substitution cipher is indeed simple in terms of its use, but it is vulnerable to elementary statistical analysis. For example, the most common ciphertext symbol corresponds to the most common plaintext symbol, which for English plaintext is most likely the letter `E`. For a homophonic substitution, the plaintext statistics tend to be flattened, since multiple ciphertext symbols can correspond to a single plaintext symbol. For example, if the plaintext is English and the letter `E` corresponds to multiple symbols, none of those symbols may stand out as having an unusually high frequency. There are several fast and effective attacks on the simple substitution, but homophonic substitutions are inherently more challenging.

Our motivation for considering homophonic substitution ciphers is the unsolved "Zodiac 340," which was created by the infamous Zodiac killer in 1969 [5]. Another Zodiac cipher, the "Zodiac 408," was a homophonic substitution that was broken within days of its publication [5]. In contrast, the Zodiac 340 has so far proved resistant to cryptanalysis. We have more to say about these ciphers in Section 6.

In this paper, we discuss the design and implementation of an efficient, general attack on homophonic substitution ciphers. We also provide extensive test results. The attack proposed here can be viewed as a generalization of the fastest known algorithm for breaking simple substitutions [7]. However, a direct generalization of the algorithm in [7] is not sufficient to solve a homophonic substitution. Therefore, we combine our generalized simple substitution algorithm with an additional hill climb layer. As with the simple substitution algorithm, only digram frequencies are used for scoring.

This paper is organized as follows. Section 2 briefly covers relevant background information. Section 3 describes a fast algorithm for solving simple substitutions [7], which we generalize to homophonic substitutions in Section 4. Then in Section 5, we present results for some of the many tests we have conducted. Section 6 covers the Zodiac ciphers and a related challenge problem, and we give the results of our attack when applied to these ciphers. Finally, Section 7 contains our conclusion and suggestions for future work.

2 Background

In this section, we consider simple substitution and homophonic substitution ciphers. We also discuss letter frequencies and the role they can play in the cryptanalysis of

these substitution ciphers. Finally, we conclude this section with a discussion of hill climbing in the context of substitution cipher cryptanalysis.

2.1 Simple Substitution

Substitution ciphers can be defined as ciphers in which every plaintext symbol has a ciphertext symbol substituted for it, and the original position of the plaintext symbol is retained in the ciphertext [10]. There are various ways in which the substitutions can be done. In this section we discuss the simplest of the substitution ciphers which, appropriately, is known as a simple substitution.

A simple substitution is a one-to-one mapping, in the sense that each plaintext symbol corresponds to one ciphertext symbol. This makes encryption and decryption straightforward, but it is well-known that the simple substitution is vulnerable to elementary statistical analysis [21]. In particular, frequency analysis can be used to attack the simple substitution. For a simple substitution, a given plaintext letter always encrypts to the same ciphertext letter and, consequently, each ciphertext letter has the same frequency in ciphertext as its corresponding plaintext letter has in the plaintext.

Suppose that the plaintext is English and only the 26 letters appear, i.e., there is no word-space, punctuation, case, etc. Then the simple substitution key space consists of the $26! \approx 2^{88}$ possible permutations of the alphabet. Consequently, the expected work for an exhaustive search is about 2^{87}.

In spite of the enormous work factor for an exhaustive search, breaking a simple substitution cipher is an elementary textbook exercise. Assuming English plaintext and a message of reasonable length, the highest-frequency ciphertext letter most likely corresponds to plaintext `E`, since `E` is the most common letter in English. This trivial observation can be extended to an effective attack by comparing English letter frequencies to ciphertext letter frequencies. Once we have determined a few of the high-frequency letters, common letter patterns become apparent and the key can then be recovered.

To develop an automated attack on a simple substitution, we must answer the following questions.

- How can we determine an initial putative key?
- How can we systematically modify the key?
- Given a putative key, how can we score the key? That is, how can we measure the "goodness" of the key?

A reasonable guess for the initial putative key can be obtained by sorting the ciphertext letters by frequency and matching the letters with the corresponding letter frequencies of English. That is, the most frequent ciphertext letter would be mapped to plaintext `E`, the second most frequent would be mapped to `T`, and so on.

To modify the key, we can simply swap pairs of letters in the putative key. Since any permutation can be obtained from another permutation by repeated swapping, this elementary approach enables us to, in principle, obtain any key.

To compute a score, we could count the number of dictionary words that appear [12]. But given a sufficiently long ciphertext, a more efficient approach is to use English digram frequencies (and, possibly, trigrams and higher-order-grams).

The major drawback to a straightforward attack on the simple substitution is that the ciphertext must be re-parsed at every iteration—for each modification to the putative key we must recompute the putative plaintext. Consequently, the work can become prohibitively expensive as the length of the ciphertext increases. We discuss a more efficient approach in Section 3, which virtually eliminates any dependence on the length of the ciphertext.

Next, we discuss homophonic substitution ciphers. In general, homophonic substitutions are much more resistant to attacks based on frequency analysis as compared to simple substitutions.

2.2 Homophonic Substitution

Instead of using a one-to-one mapping, a homophonic substitution employs a one-to-many mapping. Consequently, there may be more than one possible substitution for a given plaintext letter. However, each ciphertext symbol can represent only one plaintext letter, so the decryption is unique. As mentioned above, such a mapping tends to flatten the frequency statistics in the resulting ciphertext which makes attacks based on frequency analysis more difficult.

An example of a homophonic substitution cipher is given in Table 1. For the key in Table 1, any of the symbols `R`, `3`, or `9` can be substituted for plaintext `E`, and either `Y` or `6` can be substituted for plaintext `L`. Using this key, plaintext `HELLO` can be encrypted as `U96YB`. In this case, a cryptanalyst has no indication that ciphertext `6` and `Y` both represent the same plaintext letter. In contrast, if we encrypt `HELLO` using a simple substitution, the ciphertext would reveal that the 3rd and 4th plaintext letters are the same.

Table 1: Homophonic Substitution Key

plaintext	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
ciphertext	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
	8				9				1			6								4						
					3															5						

Suppose that a given homophonic substitution cipher has n distinct ciphertext symbols and the plaintext is English. Assuming that each of the 26 plaintext letters

maps to at least one ciphertext symbol, the theoretical key space is of size

$$\binom{n}{26} 26!\ 26^{n-26}. \tag{1}$$

Using the bound $\binom{n}{k} < n^k/k!$, equation (1) becomes

$$\binom{n}{26} 26!\ 26^{n-26} < n^{26}\ 26^{n-26} = \left(\frac{n}{26}\right)^{26}\ 26^n. \tag{2}$$

If, for example, there are $n = 52$ ciphertext symbols, then the bound in equation (2) is slightly less than 2^{271}. As another example, for $n = 104$, we find that the upper bound in equation (2) yields 2^{541}. We can also obtain a lower bound for certain cases from [22]

$$\binom{mk}{k} \geq \frac{m^{m(k-1)+1}}{(m-1)^{(m-1)(k-1)}} k^{-1/2}. \tag{3}$$

From the bound in equation (3), we find that for $n = 52$ ciphertext symbols, the keyspace is greater than 2^{258} and, consequently, for $n = 52$, we have bounded the keyspace size between 2^{258} and 2^{271}. Similarly, for $n = 104$ ciphertext symbols, we can bound the keyspace between 2^{535} and 2^{541}. Recall that for a simple substitution on 26 symbols, the key space is of size $26! \approx 2^{88}$.

In Sections 3 and 4 we consider efficient attacks on simple substitution and homophonic substitution ciphers, respectively. These attacks rely on statistical information that leaks from the plaintext into the ciphertext.

2.3 Digram Frequencies

Expected English letter frequencies are given in Figure 1. The most frequent letter in English is `E`, accounting for nearly 13% of all letters. The least frequent English letters are `Q` and `J`, each of which has an expected frequency that is much less than 1%. In contrast, for a random collection of letters, each of the 26 letters would have an expected frequency of $1/26 \approx 3.85\%$.

Digrams are consecutive pairs of symbol. For English, there are 26^2 digrams and their expected frequencies vary widely. For example, the most common English digram is `TH`, which accounts for about 1.5% of all digrams, while there are many digrams that never occur, such as `QJ`. For a random collection of letters, the expected frequency of each digram would be $1/26^2 \approx 0.15\%$.

Table 2 contains a "heat map" of English digram frequencies, that is, the cells are color-coded to indicate the frequency, with red representing higher frequencies and blue the lower frequencies [4]. Note that Table 2 includes the symbols "^" and "$" which indicate the beginning and ending of words, respectively. If we include word boundaries, the most common digram is `S$`, that is, the letter `S` at the end of a word.

Jakobsen [7] gives a fast algorithm for breaking a simple substitution, using only English digram frequencies. By searching for dictionary words (or scoring trigrams

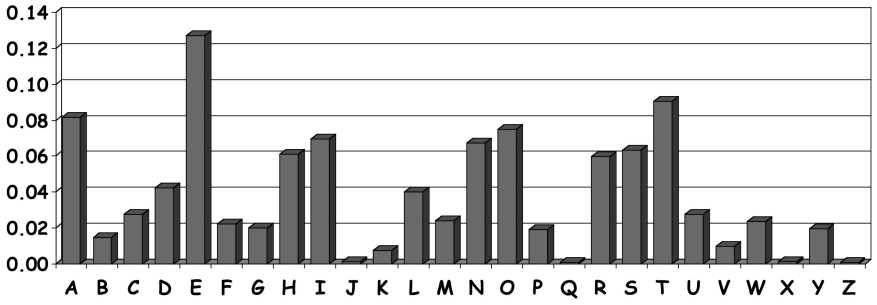
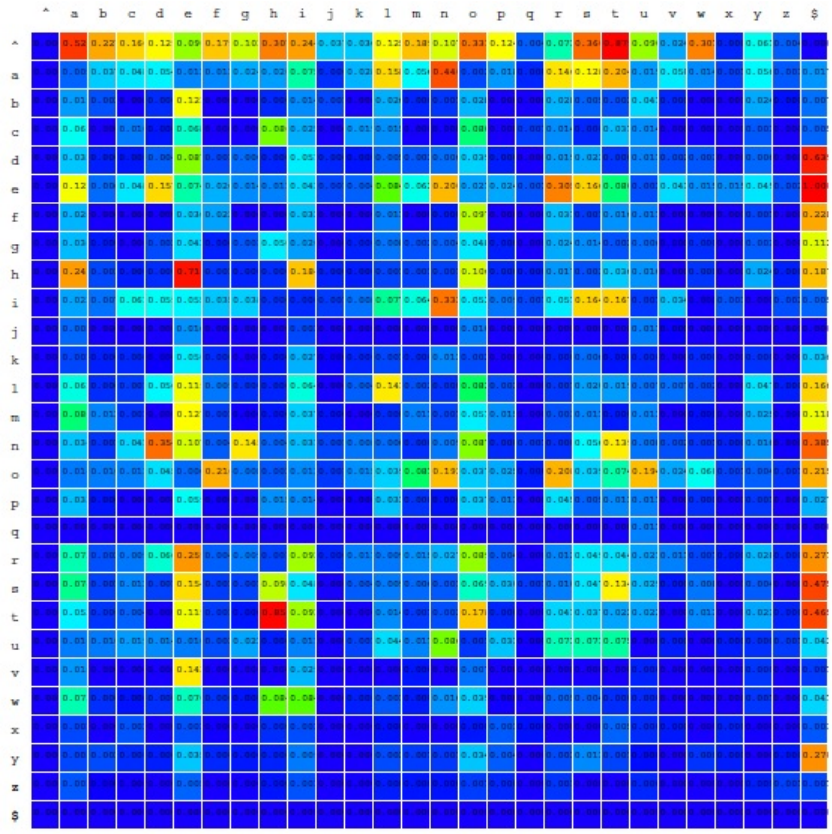


Figure 1: English Letter Frequencies

Table 2: English Digram Frequencies [4]



and, possibly, higher order "grams"), we may be able to reduce the ciphertext length requirements. However, by focusing our attention on digram frequencies, it is possible to design an extremely efficient attack. Specifically, we are able to avoid re-parsing

the ciphertext for each change in the putative key. Our fast attack on the homophonic substitution also relies only on digrams, for precisely the same reason.

We discuss Jakobsen's simple substitution algorithm in Section 3. Our generalization of this attack to the homophonic substitution is covered in Section 4. But first, we briefly discuss hill climbing in general.

2.4 Hill Climbing

Hill climbing is an an iterative technique that can be viewed as a type of heuristic search [8]. During each iteration of the hill climb, a small modification is made to the putative solution, which is then evaluated to determine whether it is an improvement over the previous solution. If the modified solution is better, the modified solution becomes the new putative solution; otherwise we make no change to the putative solution. Thus, with every iteration, the solution either improves or stays the same. That is, the algorithm can only climb "up" towards a better solution—it can never produce a worse solution. In general, a hill climb algorithm can find locally optimal solutions but, there is no assurance that it will find a globally optimal solution. In addition, hill climb algorithms tend to be very sensitive to the initial guess, so it is common to make multiple starts with different initial values.

Hill climbing works well on many substitution ciphers, including some polyalphabetic cipher machines, such as the World War II-era Japanese Purple cipher [20]. For substitution ciphers, the closer a putative key is to the actual key, the closer the corresponding putative plaintext is to the actual plaintext, which is the crucial feature that enables a hill climb (or other heuristic search) to succeed. In contrast, for strong modern ciphers, any putative key that differs from the actual key should yield putative plaintext that is uncorrelated to the actual plaintext. For example, if just one bit differs from the actual key, the resulting putative plaintext is no closer to the actual plaintext than if we had used a randomly selected key.

3 Fast Attack on Simple Substitution

In this section, we discuss Jakobsen's fast hill climb algorithm for breaking a simple substitution cipher [7]. Throughout this section we assume that the plaintext language is English, and the 26 letters of the alphabet are used as the ciphertext symbols. However, the same principles apply to other languages and, of course, we can use arbitrary symbols to represent the ciphertext.

Jakobsen's algorithm uses English digram statistics for scoring and the ciphertext is only parsed once to construct an initial digram distribution matrix. All subsequent scoring is done by directly manipulating the plaintext digram distribution matrix and then comparing the result to the expected English digram distribution matrix.

For this fast attack, we choose the initial key that best matches with English monograph statistics. That is, we assume the most frequent ciphertext letter maps

to plaintext `E`, the second most frequent maps to `T`, and so on.

At each step, we modify the putative key by swapping elements, as follows. First, denote the putative key as $K = (k_1, k_2, \ldots, k_{26})$. Then K is a permutation of the 26 letters of the alphabet. The swapping is done systematically through a series of rounds. In the first round, all adjacent elements are selected for swapping, that is, k_1 is swapped with k_2, then k_2 is swapped with k_3, then k_3 with k_4, and so on. In the second round, the elements in K at distance two are swapped, while in the third round, elements at a distance of three are swapped, and so on. Let "|" represent the swapping operation. The following series of swaps are performed:

$$\begin{array}{lllllll}
\text{round 1:} & k_1|k_2 & k_2|k_3 & k_3|k_4 & \ldots & k_{23}|k_{24} & k_{24}|k_{25} \quad k_{25}|k_{26} \\
\text{round 2:} & k_1|k_3 & k_2|k_4 & k_3|k_5 & \ldots & k_{23}|k_{25} & k_{24}|k_{26} \\
\text{round 3:} & k_1|k_4 & k_2|k_5 & k_3|k_6 & \ldots & k_{23}|k_{26} & \\
\vdots & \vdots & & \ddots & & & \\
\text{round 24:} & k_1|k_{24} & k_2|k_{25} & k_3|k_{26} & & & \\
\text{round 25:} & k_1|k_{25} & k_2|k_{26} & & & & \\
\text{round 26:} & k_1|k_{26} & & & & &
\end{array} \tag{4}$$

Whenever a swap improves the score, we update the putative key and restart the swapping algorithm from round 1. Therefore, the total number of swaps performed is not fixed, but depends on the number of times that the putative key is updated.

To fully specify this hill climb attack, we also must have an efficient method for scoring a putative key. This is accomplished by computing a score that measures the closeness of the putative plaintext digram matrix to the expected English language digram matrix. For two matrices $X = \{x_{ij}\}$ and $Y = \{y_{ij}\}$ with the same dimensions, let

$$d(X, Y) = \sum_{i,j} |x_{ij} - y_{ij}|.$$

We use this distance function to define a score as follows. Let $D = \{d_{ij}\}$ be the digram frequency matrix for the putative plaintext corresponding to a putative key K and let $E = \{e_{ij}\}$ be the expected digram frequency matrix for English. Then the score for the putative key K is defined as

$$\texttt{score}(K) = d(D, E) = \sum_{i,j} |d_{ij} - e_{ij}|. \tag{5}$$

Note that $\texttt{score} \geq 0$, with $\texttt{score} = 0$ being a perfect match, and the smaller the score, the better the key K.

The efficiency of Jakobsen's algorithm derives from the fact that when we alter the putative key by swapping elements, we can simply modify the D matrix, without re-parsing the ciphertext. When key elements are swapped, we only need to modify digram frequencies that contain the swapped elements—all other digrams are unaffected. It is easy to show that swapping elements of the putative key has the effect of

swapping the corresponding rows and columns of the D matrix, leaving the remainder of the D matrix unchanged.

To illustrate Jakobsen's algorithm, we consider a simple substitution example on the restricted 8-letter alphabet

$$\texttt{E}, \texttt{T}, \texttt{I}, \texttt{S}, \texttt{H}, \texttt{R}, \texttt{L}, \text{ and } \texttt{K}.$$

Note that these letters are listed in descending order of their expected frequencies in English. Based on this limited alphabet, suppose we are given the simple substitution ciphertext

$$\texttt{RIHILKHIERSKILEKKLKTRSKHRIHILKHLREIRTRSKLKTRSKHHLEKRS}. \tag{6}$$

For the ciphertext in (6), the frequency counts are

E	T	I	S	H	R	L	K
4	3	7	5	7	9	7	11

Consequently, our initial guess for the key is

Plaintext:	E	T	I	S	H	R	L	K
Ciphertext:	K	R	I	L	H	S	E	T

(7)

Using this initial putative key, we find the initial putative plaintext corresponding to the ciphertext in (6) is

$$\texttt{TIHISEHILTREISLEESEKTREHTIHISEHSTLITKTRESEKTREHHSLETR}$$

For this initial plaintext, the digram frequency matrix is

	E	T	I	S	H	R	L	K
E	1	1	1	2	4	0	0	2
T	0	0	2	0	0	5	1	1
I	0	1	0	3	2	0	1	0
S	4	1	0	0	0	0	2	0
H	0	1	3	2	1	0	0	0
R	4	0	0	0	0	0	0	0
L	2	1	1	0	0	0	0	0
K	0	3	0	0	0	0	0	0

(8)

The first step in Jakobsen's hill climb attack is to swap the first two elements of the key. Swapping the first two elements in (7) yields a the putative key

Plaintext:	E	T	I	S	H	R	L	K
Ciphertext:	R	K	I	L	H	S	E	T

Applying this key to the ciphertext in (6), we find that the corresponding putative plaintext is

EIHISTHILERTISLTTSTKERTHEIHISTHSELIEKERTSTKERTHHSLTER

and the digram frequency matrix is

	E	T	I	S	H	R	L	K
E	0	0	2	0	0	5	1	1
T	1	1	1	2	4	0	0	2
I	1	0	0	3	2	0	1	0
S	1	4	0	0	0	0	2	0
H	1	0	3	2	1	0	0	0
R	0	4	0	0	0	0	0	0
L	1	2	1	0	0	0	0	0
K	3	0	0	0	0	0	0	0

(9)

Comparing the matrices in (8) and (9) we see that (9) can be obtained from (8) by simply swapping the first two rows and columns—the boxed rows and columns in (9). That is, swapping letters in the putative key, recomputing the putative plaintext, and recomputing the digram matrix is equivalent to simply swapping the corresponding rows and columns of the digram matrix. This is the crucial observation that enables the fast attack to be fast since there is no need to re-parse the ciphertext. Regardless of the length of the ciphertext, only a swap of rows and columns is required.

Pseudo-code for Jakobsen's fast algorithm appears in Table 3. Note that only a single decryption of the ciphertext C is performed.

The algorithm in Table 3 is a hill climbing attack, since we ignore any modification to the key that does not improve the score. However, the result can depend on the order in which the swaps occur. Therefore, additional swaps beyond those required by the algorithm could yield improved results. We briefly return to this issue in Section 6 where we discuss attacks on the Zodiac ciphers.

In the next section, we present our fast algorithm for the homophonic substitution cipher. Our algorithm can be considered a generalization of Jakobsen's algorithm. However, the homophonic substitution presents some unique challenges.

4 Fast Attack on Homophonic Substitution

As in the previous section, we assume that the plaintext in English. However, for a homophonic substitution, multiple ciphertext symbols can represent a single plaintext letter and, consequently, we need more than 26 ciphertext symbols. For a given homophonic substitution cipher, let n be the number of ciphertext symbols. Then we have $n \geq 26$, and the special case where $n = 26$ is a simple substitution. In

Table 3: Fast Attack on a Simple Substitution Cipher [7]

```
        ////////////////////////////////////////
        // Solve simple substitution cipher:
        //     E matrix of expected plaintext digram frequencies,
        //     C is the ciphertext
        //     K = (k_1, k_2, ..., k_n) is initial putative key
        //     (listed from high to low expected frequency)
        //     d(X,Y) = sum_{i,j} |x_ij - y_ij|
        ////////////////////////////////////////
        P = putative plaintext from C using K
        D = digram distribution matrix for P
        score = d(D,E)
start:  for i = 1 to n - 1
            for j = 1 to n - i
                D' = D
                swap rows j and j + i of D'
                swap columns j and j + i of D'
                if d(D',E) < score then
                    D = D'
                    swap(k_j, k_{j+i})
                    score = d(D',E)
                    goto start
                end if
            next j
        next i
        return K
```

this section, we denote the n ciphertext symbols as $\mathtt{1}, \mathtt{2}, \mathtt{3}, \ldots, \mathtt{n}$. Furthermore, let n_a be the number of ciphertext symbols that correspond to plaintext `A`, let n_b be the number of symbols that correspond to `B`, and so on. Then

$$n_a + n_b + n_c + \cdots + n_z = n.$$

Our algorithm consists of following three "layers:"

- Inner hill climb
- Random initial key generator
- Outer hill climb

In the outer hill climb, we determine values for $n_a, n_b, \ldots, n_z$ subject to the constraint $n_a + n_b + \cdots + n_z = n$. Then in the random initial key layer, we generate multiple random initial keys, subject to the constraint given by the outer hill climb layer, namely, that the distribution of ciphertext symbols satisfy $n_a, n_b, \ldots, n_z$. Finally, for the inner hill climb we employ a generalization of Jakobsen's algorithm to hill climb to a putative key. That is, in the random key layer we determine a specific subset of n_a ciphertext symbols that map to `A`, a disjoint subset of n_b ciphertext symbols that map to `B`, and so on. Then in the inner hill climb layer, we improve on this solution, without altering any of the numeric values $n_a, n_b, \ldots, n_z$. The outer hill climb is the only layer where the numbers $n_a, n_b, \ldots, n_z$ change. These layers are nested—for each iteration of the outer hill climb, we choose multiple initial putative keys, and for each of these initial keys we conduct the inner hill climb.

Next, we discuss each of the layers in more detail, starting from the inner hill climb. Since the inner hill climb can be viewed as a generalization of Jakobsen's algorithm, we emphasize the similarities to (and differences from) the simple substitution case. After covering the inner hill climb in detail, we then discuss the random key layer and, finally, the outer hill climb. Both of these outer two layers are relatively simple.

4.1 Inner Hill Climb Layer

In this section, we assume that there are n distinct ciphertext symbols and we are given the number of these symbols that map to each plaintext letter. That is, we are given $n_a, n_b, \ldots, n_z$ with $n_a + n_b + \cdots + n_z = n$. Our goal is to develop a hill climb technique, analogous to Jakobsen's algorithm, that will, in general, yield a relatively high-scoring solution. Note that any putative solution at this phase is subject to the constraints that n_a ciphertext symbols map to `A`, n_b symbols map to `B`, and so on. That is, the values of $n_a, n_b, n_c, \ldots, n_z$ do not change at this layer. Below, we discuss the outer hill climb, where $n_a, n_b, \ldots, n_z$ are specified.

As with the simple substitution attack, we need to consider the following questions:

- How to choose the initial putative key (or keys)?
- How to modify a putative key?
- How to compute a score for a putative key?

We consider each of these questions in turn, comparing and contrasting our approach to the simple substitution attack discussed in the previous section.

Choose the initial key: From the outer hill climb layer, we have $n_a, n_b, \ldots, n_z$ with $n_a + n_b + \cdots + n_z = n$. This does not completely specify a putative key, since we still need to determine which (disjoint) subsets of $n_a, n_b, \ldots, n_z$ ciphertext symbols map to `A`, `B`, $\ldots$, `Z`, respectively. Determining such a mapping occurs in the random initial key layer, so we put off a discussion of the details of this process to Section 4.2.

Modify the putative key: Recall that for the simple substitution, we swap consecutive pairs, followed by pairs at distance two, and so on, as summarized in swapping schedule in (4), restarting each time a swap improves the score.

For the homophonic substitution, we follow an analogous process. However, in a homophonic substitution, a letter can appear multiple times in the key and we must avoid swapping such a letter with itself—swapping a letter with itself would not change the putative plaintext.

The minimum number of swaps required depends on n and $n_a, n_b, \ldots, n_z$. To simplify the notation, let $m_1 = n_a$, $m_2 = n_b$, ..., $m_{26} = n_z$. Then each of m_1 identical letters is swapped with all $n - m_1$ other letters, each of the m_2 identical letters is swapped with all remaining $n - m_1 - m_2$ letters and so on. Thus, the minimum number of score computations is given by

$$\sum_{i=1}^{26} m_i \left(n - \sum_{j=1}^{i} m_j \right) = n^2 - \sum_{i=1}^{26} m_i \sum_{j=1}^{i} m_j = \sum_{i=1}^{25} \sum_{j=i+1}^{26} m_i m_j. \tag{10}$$

In practice, we consider all $\binom{n}{2}$ pairs of ciphertext symbols, following a scheme analogous to that given in (4), and we simply skip any pairs that currently map to the same plaintext letter. The minimum number of pairs that will require a score computation is given by (10), which can be bounded above by $\binom{n}{2}$. However, as discussed above, the swapping routine restarts from the beginning whenever the score improves, so the precise number of score computations varies, depending on the ciphertext and the language statistics.

Compute the score: Recall that for the fast attack on a simple substitution, we have two matrices D and E, both of which are 26×26. The E matrix contains expected digram frequencies for English, while the D matrix contains digram frequencies corresponding to the putative plaintext. The algorithm is fast, since a swap of elements in the putative key only requires a swap of the corresponding rows and columns of the D matrix. For the simple substitution attack, we compute the score by measuring the distance between matrices D and E using the formula in (5).

For the homophonic substitution, we still use a fixed E matrix containing the expected English digram frequencies. But, instead of a single D matrix, we employ two matrices to represent the digram frequencies. Let D_C be an $n \times n$ matrix containing ciphertext digram frequencies. This matrix is constructed only once, and never changes.

The second digram distribution matrix, which we denote as D_P, is of size 26×26 and contains the letter digram frequencies corresponding to the current putative plaintext. This matrix corresponds to the D matrix in the simple substitution attack.

As in simple substitution attack, the D_P matrix is modified with each swap of elements in the putative key. However, it is not sufficient to simply swap rows and columns of D_P. Instead, we use the ciphertext digram frequencies in D_C to help determine the correct elements for the affected rows and columns of D_P. Once we

have determined D_P, the score computation is the same as in the simple substitution case, that is, we compute $\texttt{score} = d(D_P, E)$.

To illustrate the process used to update the D_P matrix, we consider a simplified example. As in the examples in Section 3, suppose that our plaintext alphabet is restricted to the eight letters

$$\texttt{E}, \texttt{T}, \texttt{I}, \texttt{S}, \texttt{H}, \texttt{R}, \texttt{L}, \text{ and } \texttt{K}.$$

Consider a homophonic substitution, where the plaintext is restricted to these eight letters, and where there are 10 ciphertext symbols, denoted $\texttt{0}, \texttt{1}, \ldots, \texttt{9}$.

Suppose that we are given the ciphertext

$$\texttt{094982495072985228610723093982480590107686107644852 07}. \tag{11}$$

Then we construct the 10×10 ciphertext digram matrix D_C,

	0	1	2	3	4	5	6	7	8	9
0	0	1	0	0	0	1	0	5	0	2
1	3	0	0	0	0	0	0	0	0	0
2	1	0	1	1	2	0	0	0	1	1
3	1	0	0	0	0	0	0	0	0	1
4	0	0	0	0	1	0	0	0	2	2
5	1	0	2	0	0	0	0	0	0	1
6	0	2	0	0	1	0	0	0	1	0
7	0	0	2	0	0	0	2	0	0	0
8	1	0	2	0	0	2	2	0	0	0
9	1	0	0	1	1	1	0	0	3	0

(12)

Next, suppose that from the outer hill climb (discussed below), we are given the constraint that two ciphertext symbols map to E and two symbols map to T, with one symbol mapping to each of the remaining six characters. With this constraint, suppose that we construct the following initial putative key:

ciphertext	0	1	2	3	4	5	6	7	8	9
plaintext	I	L	E	K	T	E	R	T	H	S

(13)

Note that the encryption version of this key can be written as

plaintext	E	T	I	S	H	R	L	K
ciphertext	2	4	0	9	8	6	1	3
	5	7						

That is, there are two ciphertext symbols that can be substituted for E (either 2 or 5) and two choices for T (either 4 or 7), while the remaining plaintext letters each have a single corresponding ciphertext value.

Applying the putative key ILEKTERTHS to the ciphertext in (11), we obtain the following putative plaintext:

ISTSHETSEITESHEEEHRLITEKISKSHETHIESILITRHRLITRTTHEEIT.

We can use this putative plaintext to generate the digram frequency matrix D_P which, in this case, is given by

$$\begin{array}{c|cccccccc} & \mathtt{E} & \mathtt{T} & \mathtt{I} & \mathtt{S} & \mathtt{H} & \mathtt{R} & \mathtt{L} & \mathtt{K} \\ \hline \mathtt{E} & 3 & 2 & 2 & 2 & 1 & 0 & 0 & 1 \\ \mathtt{T} & 2 & 1 & 0 & 2 & 2 & 2 & 0 & 0 \\ \mathtt{I} & 1 & 5 & 0 & 2 & 0 & 0 & 1 & 0 \\ \mathtt{S} & 1 & 1 & 1 & 0 & 3 & 0 & 0 & 1 \\ \mathtt{H} & 4 & 0 & 1 & 0 & 0 & 2 & 0 & 0 \\ \mathtt{R} & 0 & 1 & 0 & 0 & 1 & 0 & 2 & 0 \\ \mathtt{L} & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 \\ \mathtt{K} & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{array} \tag{14}$$

As with the simple substitution attack, at this point, we modify the putative key by swapping elements, and after each swap, we update the putative plaintext digram matrix, D_P and compute the score by comparing the updated matrix with the expected English digram matrix E. If the score improves, we update the putative key; if not, we leave the key unchanged.

For the simple substitution, updating the plaintext digram matrix D was accomplished by swapping the relevant rows and columns. For the homophonic substitution, updating D_P is slightly more involved, and we will need to make use of the matrix D_C.

Next, we illustrate the process used to update D_P by considering two examples. For the first case, suppose that in the decryption key ILEKTERTHS, we swap L and K, that is

$$\mathtt{ILEKTERTHS} \to \mathtt{IKELTERTHS}.$$

Since neither K nor L corresponds to multiple ciphertext symbols, this case is identical to swapping letters in the simple substitution. Therefore, we simply swap the K and L rows and columns of D_P, that is, we follow the same procedure as in the simple substitution attack.

The difficulty arises when one (or both) of the swapped letters correspond to more than one ciphertext symbol. For example, suppose we swap S with the first E in the putative key, that is,

$$\mathtt{ILEKTERTHS} \to \mathtt{ILSKTERTHE}. \tag{15}$$

Then we cannot simply swap the corresponding rows (and columns) or D_P, since the E row (and column) includes counts for both of the E letters that appear in the key, and we have only swapped one of the them.

Decrypting the ciphertext using the modified key in (15), we obtain the putative plaintext

IETEHSTEEITSEHESSHRLITSKIEKEHSTHIEEILITRHRLITRTTHESIT.

From this putative plaintext, we tabulate the digram frequencies to obtain the following array (the affected rows and columns are boxed):

	E	T	I	S	H	R	L	K
E	2	1	2	2	3	0	0	1
T	2	1	0	2	2	2	0	0
I	3	5	0	0	0	0	1	0
S	1	2	1	1	1	0	0	1
H	2	0	1	2	0	2	0	0
R	0	1	0	0	1	0	2	0
L	0	0	3	0	0	0	0	0
K	1	0	1	0	0	0	0	0

(16)

Note that elements outside of the E and S rows and columns are identical in (14) and (16). This is as expected, since those letters were not involved in the swapping. However, if we simply swap the E and S rows (and columns) in (14) we do not obtain the frequencies in (16). Swapping the appropriate rows and columns of (14), would give us a 0 in the HE position (i.e., in row H and column E). However, in (16) we find a 2 in the HE position. Also, swapping, would yield 0 in the upper left EE position, but in (16) we observe that EE has a count of 2.

To resolve the elements in the swapped rows and columns, we employ the D_C matrix in (12). From the putative key in (13), we note that the swapped elements E and S correspond to the rows (and columns) labeled 2 and 9, respectively. Consider the count for HE in matrix D_P. Before the swap, the H in HE corresponds to row 8 (since plaintext H is encrypted as ciphertext 8), while the E correspond to either column 2 or 5. In D_C we have a 2 in position 82 (i.e., row 8, column 2) and a 2 in position 85 giving a total of 4. This agrees with the pre-swap matrix D_P in (14), which has a 4 in position HE.

Swapping columns 2 and 9 of D_C, would put a 0 into position 82, leaving the 2 in position 85 unchanged. Therefore, post-swap, the D_P matrix will have a 2 in element HE, which agrees with (16).

In practice, we do not actually perform any swaps in D_C. Instead, we can simply examine the elements that would have been swapped. Since the key elements are swapped (in cases where the score improves), the correspondence between elements of the putative key and the rows and columns of D_C will be maintained.

The EE case mentioned above is more complex, since we need to consider four cases, and both the row and column swaps are relevant. We leave the details of this case to the reader.

The bottom line is that we can use the D_C matrix to fill in the elements in the swapped rows and columns. In the process, we do not perform any swaps on the D_C matrix itself. The elements in the putative key are swapped as appropriate, so the D_C matrix remains valid for subsequent iterations. Also, it is worth noting that there are

symmetries that can be exploited, which reduce the number of elements that must be computed using the D_C matrix by at least half.[1] Finally, note that given any putative key K, we can use the D_C matrix to determine the corresponding D_P matrix. That is, the use of D_C to compute D_P is not limited to swapping elements of the key—for arbitrary changes to the key, we can use D_C to determine all values of D_P. We make use of this fact in the random initial key layer, which we discuss next.

4.2 Random Initial Key Layer

Recall that for the simple substitution attack, we choose an initial key that gives the best match between ciphertext and English letter frequencies. Consequently, for the case of a simple substitution, it is trivial to determine an initial putative key.

For our homophonic substitution attack, the analogous step would be to chose an initial key that best matches ciphertext letter frequencies of English letter frequencies, subject to the outer hill climb constraints, $n_a, n_b, \ldots, n_z$. However, finding an optimal solution to this problem is decidedly nontrivial, since there are, in general, a large number of cases to consider. For simplicity, suppose that all plaintext letters are mapped to more than one ciphertext symbol. In this case, the number of possible initial key combinations is given by the multinomial coefficient

$$\binom{n}{n_a, n_b, \ldots, n_z} = \frac{n!}{n_a!\, n_b! \cdots n_z!}.$$

In the more general case, where some of the plaintext letters are mapped to a single ciphertext symbol, the formula is slightly more complicated. Suppose that ℓ of the plaintext letters are each mapped to a single ciphertext symbol. For these letters we can do no better than simply choosing a ciphertext symbol that most closely matches the expected English frequency. Then each of the remaining $n - \ell$ plaintext letters corresponds to more than one ciphertext symbol. Let $m_1, m_2, \ldots, m_{n-\ell}$ be the elements of the set $\{n_a, n_b, n_c, \ldots, n_z\}$ that are greater than one. The number of combinations we must consider is

$$\binom{n-\ell}{m_1, m_2, \ldots, m_{n-\ell}} = \frac{(n-\ell)!}{m_1!\, m_2! \cdots m_{n-\ell}!} \tag{17}$$

[1] When swapping putative key elements, ciphertext is conserved, that is, no characters are inserted or deleted. Consequently, any differences between the correct D_P and the results obtained by simply swapping elements (as in the simple substitution attack) must sum to zero between "swapped" pairs. For example, above we noted that if we swap the first `E` row (and column) with the `S` row (and column) in (14), then the `HE` entry would be 0. However, we see in (16) that the correct value is 2. That is, the correct value is 2 *more* than the swapped value. The corresponding element to `HE` is `HS`, that is, the values in `HE` and `HS` are swapped in the simple substitution attack. If these values were swapped, then `HS` would be 4 in D_P, but it is actually 2, which is 2 *less* than the number obtained by swapping. In general, these differences in "swap pairs" must sum to 0. So, once we compute `HE`, we can determine `HS` by reference to its swap value and `HE`, without any need to reference D_C. Similarly, `EE` is 2 greater than the value obtained by swapping, and its corresponding element of `SS` is 1, which is indeed 2 less than we would obtain by swapping.

Note that in the case of a simple substitution, $\ell = n$ and the formula in (17) yields 1, which implies that (17) holds for the simple substitution.

Since we cannot hope to choose an optimal initial key, we use a simple greedy approach. Specifically, we find subsets that approximate the letter frequencies, beginning with the highest-frequency letters and proceeding to the lower frequency letters. That is, we first find a subset of size n_e that is a reasonable approximation to the expected frequency of `E`, followed by a subset of size n_t that approximates the expected frequency of `T`, followed by a subset of size n_a that approximates the frequency of `A` in English text, and so on.

For example, suppose that $n_e = n_t = 2$ and $n_a = 1$, and we have the ciphertext frequencies in Table 4. We see that ciphertext symbols `4` and `5` can be combined to give a good approximation (12.9%) to the expected frequency of plaintext letter `E` (12.7%). Similarly, symbols `6` and `7` together approximate the expected frequency of `T` while symbol `1` approximates the expected frequency of `A`. Note that this approach will generally find good approximations for the higher frequency letters, but may not be as accurate for the lower frequency letters. This is a reasonable compromise between efficiency and accuracy, since the high frequency letters will have the largest impact on the solution.

Table 4: Initial Key

Ciphertext	`0`	`1`	`2`	`3`	`4`	`5`	`6`	$\cdots$	`n`
Frequency	8.3%	7.5%	6.7%	6.6%	6.3%	5.5%	4.4%	$\cdots$	0.1%
Initial key	`A`	`O`	`E`	`E`	`I`	`T`	`T`	$\cdots$	`Q`

As discussed in Section 2.4, a hill climb attack can only find a local maximum. One way to improve the results of any hill climb is to conduct the attack multiple times using different initial starting points. Therefore, in the random initial key layer we generate a series of R distinct initial keys, all of which are consistent with the outer hill climb layer. That is, given $n_a, n_b, \ldots, n_z$ with $n_a + n_b + \cdots + n_z = n$, for each of the R iterations of the random initial key layer, we select a subset of n_e ciphertext symbols that map to `E`, and n_t ciphertext symbols that map to `T`, and so on. For each case we use a slightly modified greedy algorithm. In practice, it is easy to find a large number of distinct initial keys.

For each of the R initial putative keys generated at this layer, we first determine the corresponding D_P (using D_C it is not necessary to re-decrypt the ciphertext), then complete the inner hill climb. The final result of the random initial key layer is the best-scoring putative key obtained in any of these R iterations over the inner hill climb.

We conducted experiments to empirically determine a useful value for R. These experiments are discussed in Section 5.

4.3 Outer Hill Climb

In Section 4.2 we discussed the method used to generate initial keys that satisfy the constraint $n_a, n_b, \ldots, n_z$ where $n_a + n_b + \cdots + n_z = n$. The outer hill climb layer provides these constraints, that is, the outer hill climb specifies the number of ciphertext symbols that are mapped to each letter.

As the name indicates, this layer uses a hill climb approach. That is, we initially specify $n_a, n_b, \ldots, n_z$ with $n_a + n_b + \cdots + n_z = n$. Then we iterate, where at each iteration we modify these subset sizes by "swapping" adjacent pairs (ranked from high frequency to low). The swapping is somewhat different than in the inner hill climb, so we explain it in more detail below. But, the point is that for each swap we compute a score (i.e., the best score obtained via the random key and inner hill climb layers) and use this score to guide adjustments in the distribution of letters to symbols. That is, the goal of the outer hill climb is to climb to the correct *distribution* of ciphertext symbols to letters.

First, we need to specify an initial distribution to begin the outer hill climb. We begin by, in a sense, assuming the worst case. That is, we assume that the mapping of letters to ciphertext symbols was chosen to flatten the ciphertext statistics as much as possible. Consequently, we assume that about 12% of the symbols correspond to `E`, while the letter `T` corresponds to about 9% of the symbols, and so on. In addition, we assume that at least one symbol corresponds to each letter of the alphabet. Examples of such initial distributions for various ciphertext symbol sizes appear in Table 5.

Table 5: Initial Frequency Distribution

n	n_e	n_t	n_a	n_o	n_i	n_n	n_s	n_r	n_h	n_d	n_l	n_c	n_u	n_m	n_f	n_w	n_g	n_y	n_p	n_b	n_v	n_k	n_x	n_j	n_q	n_z
26	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
27	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
35	4	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
45	5	4	3	3	3	3	3	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
55	7	5	4	4	4	3	3	3	3	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
65	8	6	5	5	5	4	4	4	4	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1
75	9	7	6	6	5	5	5	4	4	3	3	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1
85	11	8	7	7	6	6	5	5	5	3	3	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1
95	12	9	8	7	7	7	6	6	5	4	4	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1

Next, we discuss the outer hill climb "swapping" process in more detail. Suppose that $n = 35$. Then for the initial distribution, as given in Table 5, we have $n_e = 4$ and $n_t = 2$ and so on. Using this distribution, we execute the random initial key layer which, in turn, calls the inner hill climb. The best score of this entire process is saved as the current `score`. Then we "swap," starting with n_e and n_t. However,

instead of simply swapping the values, we increment n_e and decrement n_t. So, for this example, the first swap consists of setting $n_e = 5$ and $n_t = 1$, with the remaining values unchanged. Note that the distribution still satisfies $n_a + n_b + \cdots + n_z = n$, as required. Using this new distribution, we call the random initial key layer, which calls the inner hill climb. If the best score from this process is less than `score`, then we update `score` and maintain this new distribution; if not, `score` is unchanged and we revert to the previous distribution.

If the score does not improve, then we switch the order of the pair and "swap" again. For the example in the previous paragraph, we would increment the original n_t and decrement the original n_e. That is, we let $n_t = n_e = 3$ and execute the inner layers. Then for the next swap, we consider the pair n_t and n_a, and so on.

We first perform this outer swap on all consecutive pairs, then all pairs at distance 2, then all pairs at distance 3, and so on. The process is analogous to that used in the inner hill climb and illustrated in (4). However, in this case, the order matters, since one element is incremented and the other decremented, so we have twice as many cases to consider. Again, we do not swap the actual values, but instead, we increment and decrement the counts, subject to the constraint that every count must be at least 1. Since every count must be at least 1, whenever the count to be decremented is 1, we skip that step.

4.4 All Together Now

A high level overview of the algorithm appears in Figure 2. This diagram illustrates the relationship between the three layers. Pseudo-code for the attack appears in Tables 6, 7 and 8.

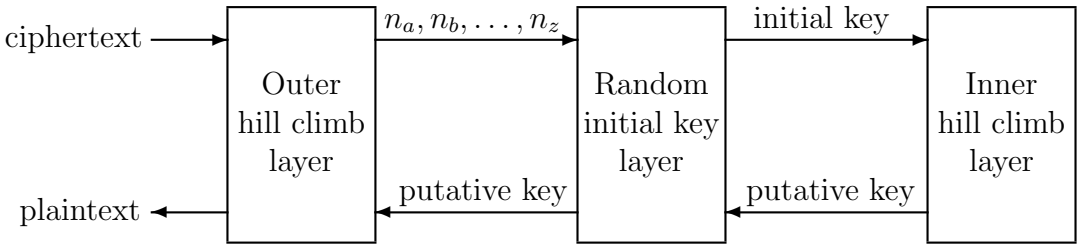


Figure 2: Block Diagram

For the outer hill climb layer, the "outerSwap" function is defined as

outerSwap(m_i, m_j)
    increment(m_i)
    decrement(m_j)
end outerSwap

Table 6: Outer Hill Climb Layer

```
OuterHillClimb
global K = bestInitKey = bestKey = NULL
parse ciphertext to determine D_C
initialize n_a, n_b, ..., n_z as in Table 5
(m_1, m_2, ..., m_26) = (n_a, n_b, ..., n_z)
bestScore = RandomInitialKey(m_1, m_2, ..., m_26)
bestKey = bestInitKey
for i = 1 to 25
    for j = 1 to 26 − i
        (m'_1, m'_2, ..., m'_26) = (m_1, m_2, ..., m_26)
        outerSwap(m'_j, m'_{j+i})
        score = RandomInitialKey(m'_1, m'_2, ..., m'_26)
        if score < bestScore then
            (m_1, m_2, ..., m_26) = (m'_1, m'_2, ..., m'_26)
            bestScore = score
            bestKey = bestInitKey
        else
            (m'_1, m'_2, ..., m'_26) = (m_1, m_2, ..., m_26)
            outerSwap(m'_{j+i}, m'_j)
            score = RandomInitialKey(m'_1, m'_2, ..., m'_26)
            if score < bestScore then
                (m_1, m_2, ..., m_26) = (m'_1, m'_2, ..., m'_26)
                bestScore = score
                bestKey = bestInitKey
            end if
        end if
    next j
next i
return bestKey
```

To simplify the pseudo-code for the outer hill climb that appears in Table 6, we assume that the second argument to outerSwap is greater than one. If not, we skip that particular step, since we require that at least one ciphertext symbol maps to each plaintext letter, that is, $m_i \geq 1$ for all i.

Table 7: Random Initial Key Layer

```
RandomInitialKey(n_a, n_b, ..., n_z)
bestInitScore = ∞
for r = 1 to R
    randomly initialize K = (k_1, k_2, ..., k_n) satisfying n_a, n_b, ..., n_z
    D_P = digram matrix from D_C and K
    initScore = InnerHillClimb(D_P)
    if initScore < bestInitScore then
        bestInitScore = initScore
        bestInitKey = K
    end if
next r
return bestInitScore
```

Table 8: Inner Hill Climb Layer

```
InnerHillClimb(D_P)
innerScore = d(D_P, E)
for i = 1 to n − 1
    for j = 1 to n − i
        K' = K
        swap(k'_j, k'_{j+i})
        D' = digram matrix for K' using D_P and D_C
        if d(D', E) < innerScore then
            innerScore = d(D', E)
            K = K'
            D_P = D'
        end if
    next j
next i
return innerScore
```

4.5 Work Factor

The size of the keyspace for a simple substitution and homophonic substitution are discussed in Sections 2.1 and 2.2, respectively. Here, we consider the work factor for the fast simple substitution and homophonic substitution attacks.

Recall that for the simple substitution attack—and for the inner hill climb in the homophonic substitution attack— we restart from the beginning of the swap schedule each time the score improves. This makes the analysis of the average work factor difficult to compute. For the simple substitution attack, we can say that in the best case, the work factor is $\binom{26}{2} = 325$ score computations.

Determining the work factor for the homophonic substitution is extremely challenging, even if we consider a best-case analysis, and such results would be of little value in determining the average performance. Therefore, we simply provide empirical results for both the simple substitution and homophonic substitution.

We find that as we increase the size of the ciphertext message, the simple substitution attack approaches our best-case approximation. These results are summarized in Table 9. Note that these results are based on 10 test cases for each ciphertext length.

Table 9: Score Calculation for Simple Substitution Attack

ciphertext length	500	1000	3000	5000	8000
score computations	1050	1010	905	790	630

Empirical results for the homophonic substitution attack appear in Table 10. Note that these results are based on sample sizes of 3 test cases each. For an alphabet of size 55, the homophonic substitution attack is slower than the simple substitution attack by a factor of about 20 to 50, depending on the ciphertext length. As expected, for a smaller alphabet size, the homophonic attack is more competitive with the simple substitution attack.

Table 10: Score Calculation for Homophonic Substitution Attack

	ciphertext length				
alphabet size	500	1000	3000	5000	8000
35	9230	8990	10,370	11,550	11,710
45	14,980	15,520	17,760	20,450	19,770
55	23,750	24,730	27,380	35,170	30,800

5 Experimental Results

We have implemented the fast homophonic substitution attack discussed in the previous section. The code was written in C++ in a UNIX environment.

The test cases discussed in this section cover different scenarios involving various ciphertext sizes, number of ciphertext symbols, and number of initial starting points provided to the inner hill climb layer. For all test cases, the plaintext is English and hence the E matrix contains standard English digram statistics.

5.1 Simple Substitution Results

In this section, we briefly consider test cases where our algorithm is applied to a simple substitution cipher. For each test, we used our inner hill climb algorithm and the random initial key layer with $R = 40$ initial starting points. Note that this algorithm is essentially equivalent to 40 iterations of Jakobsen's algorithm, with random initial starting points in each iteration. The plaintext consists of 27 symbols (26 letters and space), and the following ciphertext length were considered: 300, 500, 700, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, and 10,000 characters. The graphs in Figure 3 summarize the results for these test cases.

From the graphs in Figure 3, we clearly see that as the length of the ciphertext increases, the score improves and the percentage of correct symbols increases. With a ciphertext length of 500 or more, we can easily recover the plaintext. These results are as expected.

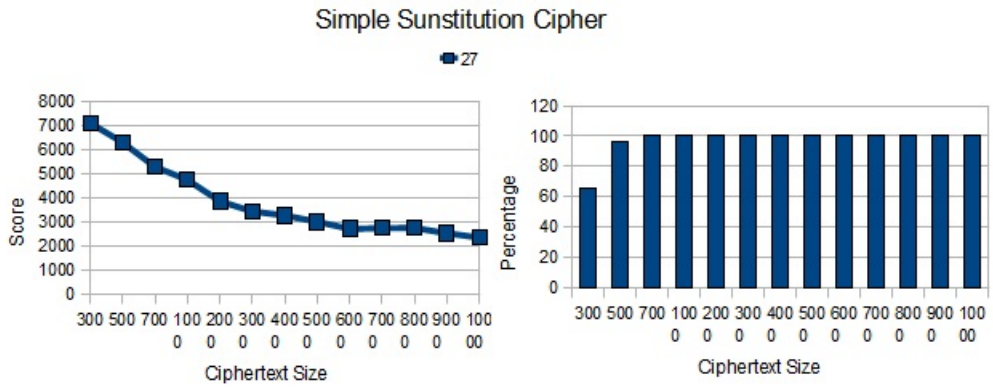


Figure 3: Simple Substitution Cipher

5.2 Homophonic Substitution Results

We applied our attack code to more than 1000 homophonic substitution test cases. For these test cases, ciphertext lengths ranged from 300 to 10,000 characters, cipher alphabet sizes ranged from 27 to 100, and the numbers of initial starting points ranged from 1 to 100. For all test cases reported here, the plaintext consists of 27 symbols (26 English letters plus the space).

In the remainder of this section, we summarize several test cases. These test cases only present a small subset of the tests that were conducted.

- **Case 1**: In this case, we test the inner hill climb layer. To do so, we assume that the outer hill climb generated the correct letter distribution, that is $n_a, n_b, \ldots, n_z$ exactly matches the actual key. Also, we use one initial random starting point.
  - Number of ciphertext symbols: 28, 35, 45, 55, and 63.
  - Ciphertext lengths: 300, 500, 700, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, and 10,000
  - Results: The graph in Figure 4 presents the success rates. Note that with a success rate of about 80% or more, it is easy to manually recover the correct plaintext and key.
  - Observations: As expected, the solutions improve as the length of the ciphertext increases, regardless of the number of ciphertext symbols. However, with limited ciphertext, the results are poor and inconsistent.
- **Case 2**: This test case is the same as Case 1, except that we use 40 initial random starting points. That is, Case 1 tested the inner hill climb layer alone, while this case tests the inner hill climb and random initial key layers together.
  - Number of ciphertext symbols: 28, 45, 65, 85, and 100.
  - Ciphertext lengths: 300, 500, 700, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, and 10,000
  - Results: The graph in Figure 5 gives the success rates.
  - Observations: Including the random initial key layer greatly improved the results. Also, from the graph is Figure 5, it appears that for homophonic substitutions with 100 or more cipher symbols, our results are likely to be poor, regardless of the length of the ciphertext message.
- **Case 3**: Here, we duplicate much of the previous test case, but use 100 random starting points for the random initial key layer—as opposed to 40 in Case 2.
  - Number of ciphertext symbols: 28, 35, 45, 55, and 65.
  - Ciphertext lengths: 300, 500, 700, and 1000
  - Results: The graph in Figure 6 contains the success rates.
  - Observations: The results are not significantly improved over the previous test case, so we conclude that 40 initial starts is sufficient.
- **Case 4**: Inner hill climb layer with random initial key layer (with 40 initial starting points per iteration) and outer hill climb layer. Note that this is the first test of our complete algorithm.
  - Number of ciphertext symbols: 28, 35, 45, 55, and 65
  - Ciphertext lengths: 300, 500, 700, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, and 10,000

– Results: The graph in Figure 7 presents the success rates for this case.
– Observations: Including the outer hill climb layer gives us similar results to Case 2. This is a very positive finding since, in effect, Case 2 assumes the optimal output from the outer hill climb layer. The results for this test case indicate that the outer hill climb layer is effective.

Finally, we summarize the results for our homophonic substitution attack in the form of the 3-dimensional graph in Figure 8. The graph shows the success rate (z-axis) as a function of the ciphertext length (x-axis), and number of ciphertext symbols (y-axis). A success rate of about 80% is sufficient to easily recover the plaintext

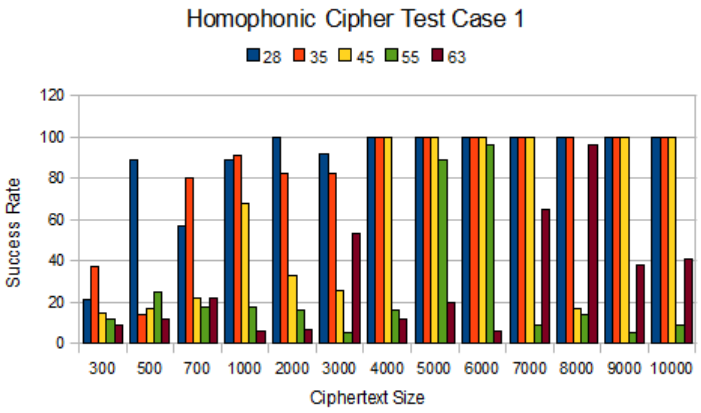


Figure 4: Case 1 Success Rates

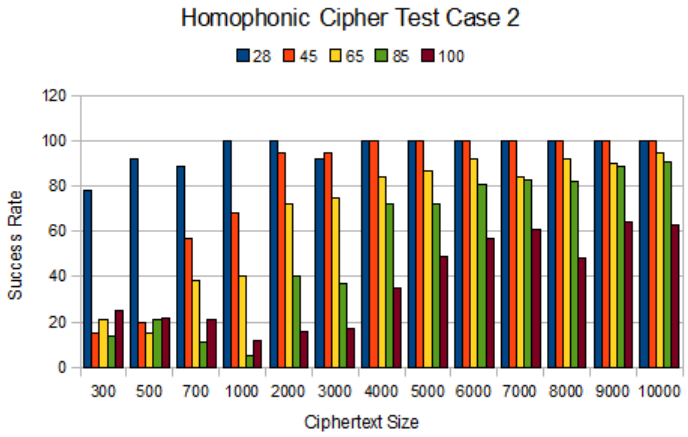


Figure 5: Case 2 Success Rates

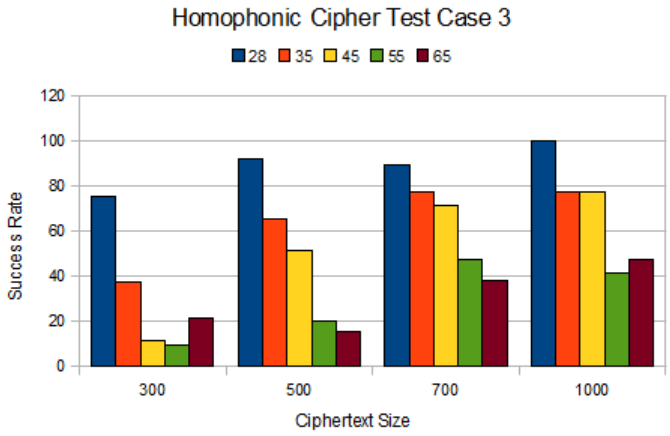


Figure 6: Case 3 Success Rates

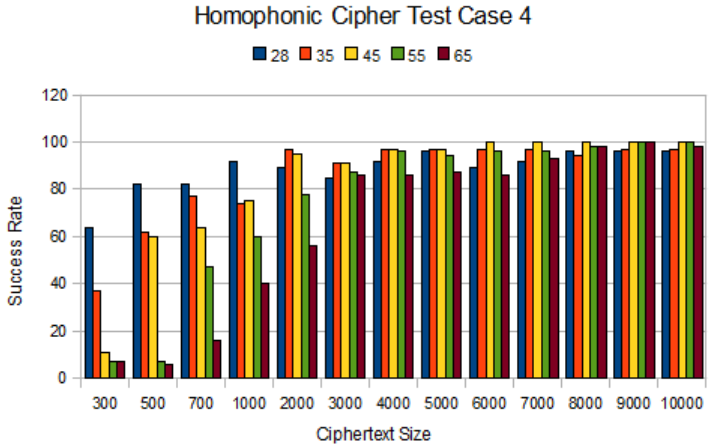


Figure 7: Case 4 Success Rates

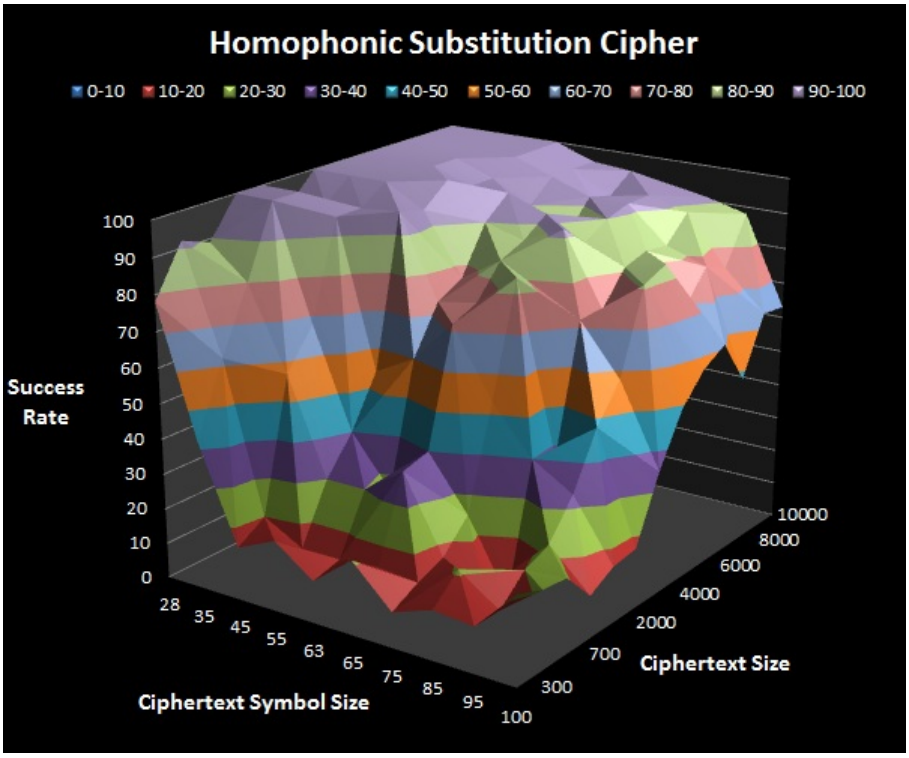


Figure 8: Homophonic Substitution Success Rates

6 Zodiac Ciphers

The Zodiac was a serial killer who claimed at least seven victims (five dead, two injured) in the San Francisco vicinity during the late 1960s and early 1970s [5]. Nobody was prosecuted for the Zodiac killings. During his killing spree, the Zodiac sent letters to local newspapers and police, often taunting police for their failure to determine his identity. Some of these letters included ciphers and apparent ciphers.

The "Zodiac 408" cipher—so-called because it has 408 ciphertext symbols—was a three-part ciphertext, with one part sent to the *San Francisco Chronicle*, one part sent to the *San Francisco Examiner*, and the third part sent to the *Vallejo Times-Herald*. The cipher, which is a homophonic substitution, was broken within six days of its publication by local school teachers, Donald and Bettye Harden [1].

In addition to the Zodiac 408 cipher, the Zodiac killer sent three more messages that appear to be ciphertext. Two of these are very brief and therefore not amenable to cryptanalysis. However, the unsolved "Zodiac 340" (ciphertext of 340 symbols) appears to offer some prospects for cryptanalysis.

Next, we discuss the solved Zodiac 408 cipher. Then we consider the Zodiac 340 and a related cipher challenge problem.

6.1 Zodiac 408 Cipher

The Zodiac 408 cipher is displayed in Figure 9. Note that each of the three parts of the ciphertext message is displayed as eight rows of 17 symbols.

The solution to the Zodiac 408 cipher is given in Table 11, where we display two rows of the original message per line, with a small gap between the two. Also, we have left a gap between each of the three parts.

Note that the misspellings appear in the original message. Some of these "misspellings" might be due to the fact that a thick-tipped pen was used, and several of the symbols are similar. Also, the final 18 symbols appear to be filler used to make the cipher fit a predetermined grid [6].

The frequency distribution for the Zodiac 408 plaintext is given in Table 12. The letters `J`, `Q`, and `Z` do not appear in the plaintext.

6.2 Zodiac 340 Cipher

The unsolved Zodiac 340, which appears in Figure 10, is reminiscent of the Zodiac 408. This apparent ciphertext was mailed to the *San Francisco Chronicle* on November 8, 1969 [16]. There have been many proposed solutions to the Zodiac 340, but none of the "solutions" published to date holds up under scrutiny. For recent examples of proposed solutions, see [18] or [23]. The website [14] gives a good analysis of the flaws in the supposed solution [18].

It is possible that the Zodiac 340 is a homophonic substitution combined with another encryption technique. Of course, another possibility is that the Zodiac 340 is

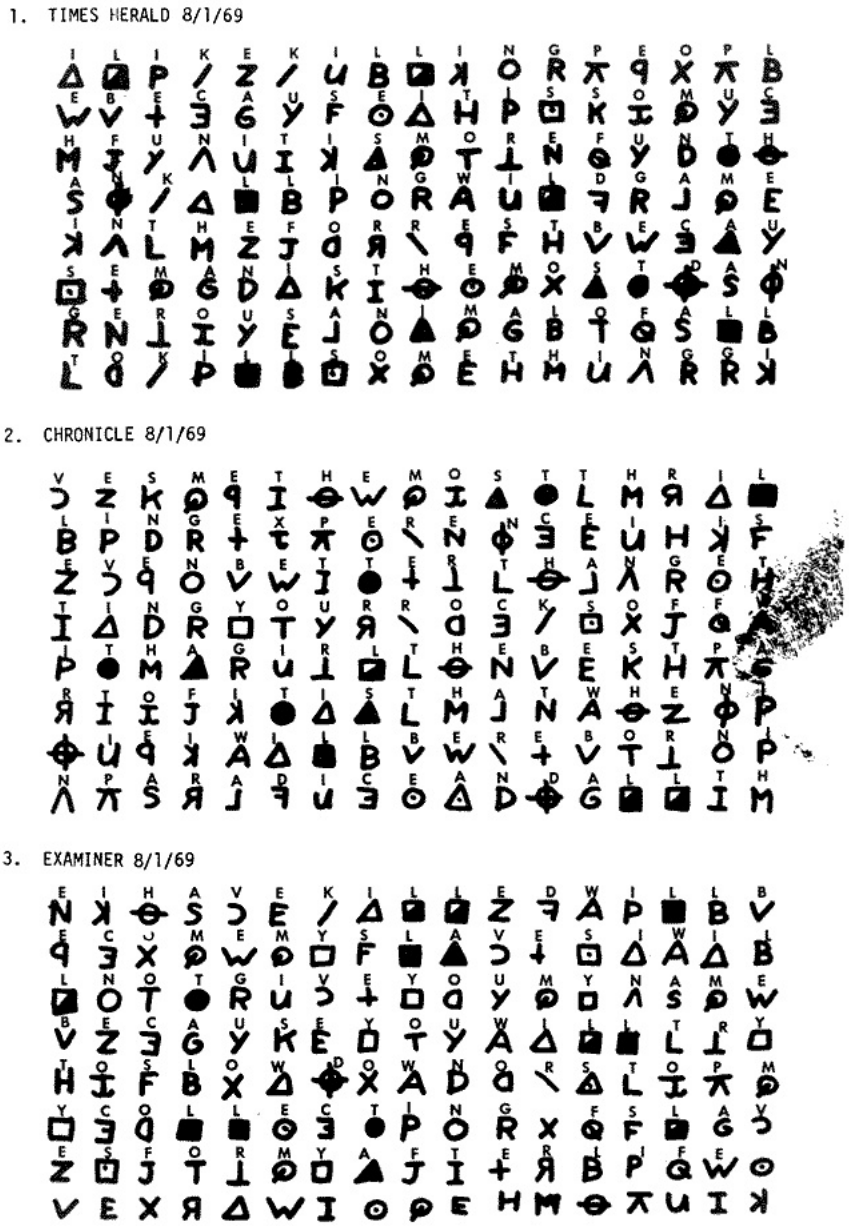


Figure 9: Zodiac 408 Cipher [19]

not a ciphertext—it could be nothing more than a random collection of symbols. For the remainder of this paper, we assume the Zodiac 340 is a homophonic substitution.

We conducted several tests on the Zodiac 340 cipher. Two typical test cases are summarized below.

Table 11: Zodiac 408 Plaintext

```
ILIKEKILLINGPEOPL EBECAUSEITISSOMUC
HFUNITISMOREFUNTH ANKILLINGWILDGAME
INTHEFORRESTBECAU SEMANISTHEMOSTDAN
GEROUSANAMALOFALL TOKILLSOMETHINGGI

VESMETHEMOSTTHRIL LINGEXPERENCEITIS
EVENBETTERTHANGET TINGYOURROCKSOFFW
ITHAGIRLTHEBESTPA RTOFITISTHAEWHENI
DIEIWILLBEREBORNI NPARADICEANDALLTH

EIHAVEKILLEDWILLB ECOMEMYSLAVESIWIL
LNOTGIVEYOUMYNAME BECAUSEYOUWILLTRY
TOSLOIDOWNORSTOPM YCOLLECTINGOFSLAV
ESFORMYAFTERLIFEE BEOROETEMETHHPITI
```

Table 12: Zodiac 408 Plaintext Frequency Counts

letter	E	I	T	L	O	S	A	N	R	M	H	G	F	U	C	Y	W	P	B	V	K	D	X
count	52	40	35	31	26	23	23	22	16	16	16	12	11	10	10	8	7	7	7	6	6	6	1

- **Case 1**: Inner hill climb with 40 random starts (no outer hill climb)
  - Plaintext symbols: 26 letters (no space)
  - Score: 6092
  - Putative plaintext:

```
ssunolldfrhaemert totsthiveiwaneryh
sieagtebeithehast dooilloedroitturr
ongtanttheeftfors ooulelicetorenret
hetrkvareapedaaeh tertythitotwatzen
tngtxtburesaiqjor caledtlttioinesic
etusenshewsoxmgri rnalutheemegatfse
ajndihttheesataou leetlyerexaadentt
ispaeeonosndebect haarasrtiolaygker
uamteerontelishhe tthioaiafidsmstho
hertininsowisecoa ngestordabioqhrat
```

- **Case 2**: Outer hill climb with inner hill climb and 40 random starts
  - Plaintext symbols: 26 letters (no space)

Figure 10: Zodiac 340 Cipher [16]

– Score: 5868
– Putative plaintext

```
passturinytistref otinatlkealdethen
ngidwineraanstmpb atthruetiqtliishe
cowimerfthonintys tesrouletfryoseea
thihakvendoridmst arebertlicolmizso
fewiaieseondgxjth emuniirbiltheonge
nissheatelntatwhl eedusbtththwdinno
mjoagtiothondaves uroorehehaddiheii
```

```
gnoititorasinehei tddqmnhaltrdeware
sdtishereiouantns iinarmadnlaptsant
toyfaehencllnterd swepotyamelexthvi
```

Obviously, neither of these solutions is particularly impressive. Also, neither score is indicative of a strong match to English.

6.3 MysterTwister Zodiac Challenge

The MysteryTwister website [11] includes a large number of interesting cipher challenges, ranging from pencil-and-paper to research-level problems. One of the challenges posted on the MysterTwister website was inspired by the Zodiac cipher. We applied our homophonic substitution attack on this challenge problem. Note that the challenge problem ciphertext has 340 character, with 65 distinct symbols. That is, the challenge was designed to mimic the Zodiac 340 cipher.

We consider two cases. First, we use English digrams in the target matrix E. That is, we follow the same procedure as used in all of the examples presented to this point. Second, we consider a case with the Zodiac 408 plaintext as the expected "language," that is, we replace the E matrix with the digram statistics of the Zodiac 408 plaintext. Since the Zodiac 408 is known to be the Zodiac's writing, this might better model the underlying plaintext than standard English. The results for these two test cases are summarized below.

- **Case 1**
  - Outer hill climb layer with initial random key layer (40 initial starts per iteration) and inner hill climb layer
  - E matrix: English digrams
  - Score: 5567
  - Putative plaintext:

```
orevcochinnoasabe xdeteoghetrytieje
alounleytansismay ankerirntiorftdth
onaholatthgldotes theduosthspattfen
oenassaueperandee livehishectyemoon
segecahetigatyree rontoqqhtsutheles
frioichedcridarme mbendfothanfarily
oeadsconereficehd ithecprgiassseiee
imaloesorhotrmdte dhedstorisineratr
tagrhifainingthap rearestlomthlshes
otniterdithrroncc dcanretopothaqclr
```

- **Case 2**
  - Outer hill climb layer with initial random key layer (40 initial starts per iteration) and inner hill climb layer

– E matrix: Zodiac 408 plaintext digrams
– Score: 4229
– Putative plaintext:

```
ilidetillingreopl dbecaeseittisomal
ezenitiimofeaunth anjilltngwildgame
inthezorrestbecau semanisthenostdan
gefousaninalosall todillsomethinggi
vesmethemostthril lingexqerenceitis
dteiwillbereborni npafadiceandallth
eieavetilledwillb ecomenrslavesiwil
lnotgiveroemrname belauserouwilltrr
toslowdownofstorn rlollectingozslav
essormraaterlisee beoftetenetheqitt
```

From the putative plaintext obtained in Case 2, we can complete the solution manually. The solution is given by (with spaces added):

> I like killing people because it is so much fun it is more fun than killing wild game in the forrest because man is the most dangerous animal of all to kill something gives me the most thrilling experence it is die I will be reborn in paradice and all the I have killed will become my slaves I will not give you my name because you will try to slow down or stop my collecting of slaves for my afterlife ebeorietemethhpiti

As an additional experiment, we modified the outer hill climb so that it does a more thorough, but possibly slower, hill climb. In this modified hill climb, we "swap" all adjacent pairs of elements, and then iterate, that is, we again swap all adjacent pairs of elements, and again, and so on. This continues until we complete one entire iteration without any swap improving the score. We refer to this as the "slow" outer hill climb, as opposed to our standard outer hill climb which we call the "fast" approach, although it is possible that in some cases, the "fast" hill climb may be slower than the "slow" hill climb, and vice versa. Our test results comparing the slow versus fast outer hill climb are given in Table 13.

Table 13: MysteryTwister Zodiac Challenge

Case	Description	Percentage
1F	Fast outer hill climb and English stats	13.00%
1S	Slow outer hill climb and English stats	4.00%
2F	Fast outer hill climb and Zodiac 408 stats	70.00%
2S	Slow outer hill climb and Zodiac 408 stats	84.00%

From Table 13 we again see that using the Zodiac 408 statistics for the E matrix gives much better results than standard English statistics. However, it is interesting to note that the slow hill climb gives significantly better results when the Zodiac 408 statistics are used, while it actually yields worse results when English statistics are used. The explanation appears to be that the slow hill climb does indeed yield "better" results, in the sense of producing putative plaintext that more closely models the target matrix E. However, the Zodiac challenge cipher does not fit the standard English model particularly well. Consequently, as the hill climb better matches the English matrix E, it match the true plaintext less. In contrast, using the Zodiac 408 statistics (which are virtually identical with those of the Zodiac challenge), a stronger match to the E matrix, as in Case 2S of Table 13, produces more accurate results.

The results in this section strongly indicate that our algorithm is very sensitive to the language model (as represented by the E matrix). Therefore, it might be worthwhile to consider specialized "language" models when trying to break the Zodiac 340. It might also be useful to employ the slow outer hill climb, but that appears to yield a relatively modest improvement. However, for the Zodiac 340, we have a brief ciphertext (only 340 symbols) and a relatively large alphabet (62 symbols). From Figure 8, we see that for these parameter values, we will need every possible advantage to have a reasonable chance of breaking the cipher—assuming it actually is a homophonic substitution.

7 Conclusions and Future Work

We designed and implemented an efficient attack on homophonic substitution ciphers. The attack utilizes a nested hill climb approach, and can be viewed as a generalization of the fastest known attack on simple substitution ciphers. The algorithm was implemented and extensively tested. We were able to consistently recover at least 80% of the plaintext characters for ciphers having 42 ciphertext symbols or less, provided we have a ciphertext of 1000 or more characters. If the ciphertext length is at least 3000 characters, then we achieve the same level of success for ciphers with 75 or fewer ciphertext symbols.

These results indicate that the Zodiac 340 cipher—assuming it is a homophonic substitution—may be out of range for our attack. However, tests indicate that with a more accurate plaintext "language" model and a more thorough outer hill climb, we should have a realistic chance of breaking a cipher such as the Zodiac 340. Therefore, our best chance of success is, perhaps, to devise a strong language model for the unknown plaintext. This would require some degree of insight and luck.

There are several possible improvements to our attack algorithm. The slow outer hill climb that was briefly discussed in Section 6.3 is one such improvement. Another possible modification to the outer hill climb is to make multiple "swaps" with a given pair until the score no longer improves, before moving on to the next pair. However, these modifications would significantly increase the cost of the algorithm.

The random initial key layer is another area where alternative approaches could improve the overall attack. The greedy approach used at this layer appears to make the algorithm particularly sensitive to the expected statistics. This is a good feature in cases where the plaintext statistics are known, but could be detrimental in cases where the statistics are less certain.

It might seem natural to extend the scoring to include trigrams and, possibly, higher order "grams." However, even for trigrams, swapping and scoring would both become extremely complex, and the work factor would increase exponentially. Dictionary-based attacks also seem tempting, but have similar drawbacks.

A potentially profitable modification to our algorithm would be to include a "crib" feature. That is, the user could specify a word that might appear in the plaintext, and this word could be tried at every offset in the ciphertext. This would be a straightforward modification.

Various heuristic search techniques could be combined with the general approach discussed in this paper. For example, the outer hill climb could be replaced by a genetic algorithm [8]. While genetic algorithms have been applied to homophonic substitution ciphers [3, 13], they have not been used in a manner similar to the method suggested here. Other heuristic search techniques could also be considered.

References

[1] A 'murder code' broken, *San Francisco Chronicle*, August 9, 1969, `http://www.flickr.com/photos/seanutbutter/2462841231/`

[2] Announcing the Advanced Encryption Standard (AES), NIST, FIPS 197, November 26, 2001, `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`

[3] P. Basavaraju, Heuristic-search cryptanalysis of the Zodiac 340 cipher, Master's report, Department of Computer Science, San Jose State University, 2009, `http://www.cs.sjsu.edu/faculty/stamp/students/kanagalakatte_pallavi.pdf`

[4] K. Briggs, English and Latin digram and trigram frequencies, `http://keithbriggs.info/documents/english_latin.pdf`

[5] G. Claston, 340-cipher — overview and examination, `http://www.zodiackiller.com/mba/zc/69.html`

[6] Glurk, The final 18 symbols are nothing but filler!, `http://www.zodiackillerfacts.com/forum/viewtopic.php?f=49&t=423`

[7] T. Jakobsen, A fast method for the cryptanalysis of substitution ciphers, *Cryptologia*, Vol. 19, pp. 265–274, 1995

[8] D. Kreher and D. Stinson, *Combinatorial Algorithms: Generation, Enumeration and Search*, CRC Press, 1998

[9] Local Obstacle Avoidance and hill-climbing, GamePlayDev, http://www.gameplaydev.com/2010/08/local-obstacle-avoidance-and-hill-climbing/

[10] J. Mathai, History of computer cryptography and secrecy system, http://www.dsm.fordham.edu/~mathai/crypto.html

[11] MysteryTwister C3: The crypto challenge contest, http://www.mysterytwisterc3.org/

[12] E. Olson, Robust dictionary attack on short simple substitution ciphers, *Cryptologia*, Vol. 31, No. 4, pp. 332–342, 2007, http://april.eecs.umich.edu/pdfs/olson2007crypt.pdf

[13] D. Oranckchak, Evolutionary algorithm for decryption of monoalphabetic homophonic ciphers encoded as constraint satisfaction problems, *GECCO '08*, July 12–16, 2008, http://oranchak.com/t14pap379-oranchak.pdf

[14] D. Oranckchak, Corey Starliper claims to have solved the 340-character Zodiac Killer cipher, http://oranchak.com/zodiac/corey/hoax.html

[15] P. C. Pop and I. Zelina, Heuristic algorithms for generalized minimum spanning tree problem, *Proceedings of ICTAMI 2004*, http://www.uab.ro/auajournal/acta8/Pop-Zelina.pdf

[16] Preliminary report on project MK-ZODIAC, 2011, http://mk-zodiac.com/game.html

[17] ROT 13 — simple substitution cipher, http://www.tech-faq.com/rot-13.html

[18] B. Schillemat, Has the code of the Zodiac killer been cracked?, *Forest City Patch*, http://fostercity.patch.com/articles/has-the-code-of-the-zodiac-killer-been-cracked

[19] Solved Zodiac 408 cipher, http://wiki.zodiac-ciphers.dreamhosters.com/wiki/Solved_408-character_cipher

[20] M. Stamp and R. M. Low, *Applied Cryptanalysis: Breaking Ciphers in the Real World*, Wiley, 2006

[21] M. Stamp, *Information Security: Principles and Practice*, second edition, Wiley, 2011

[22] P. Stănică, Good lower and upper bounds on binomial coefficients, *Journal of Inequalities in Pure and Applied Mathematics*, Vol. 2, Issue 3, Article 30, 2001, http://www.emis.de/journals/JIPAM/images/043_00_JIPAM/043_00.pdf

[23] The Zodiac 340 cipher solved, http://www.opordanalytical.com/articles1/zodiac-340.htm