Evolving Keys for Periodic Polyalphabetic Ciphers

Ralph Morelli and Ralph Walde
Computer Science Department
Trinity College
Hartford, CT 06106
ralph.morelli@trincoll.edu

Abstract

A genetic algorithm is used to find the keys of Type II periodic polyalphabetic ciphers with mixed primary alphabets. Because of the difficulty of the ciphertext only cryptanalysis for Type II ciphers, a multi-phased search strategy is used, each phase of which recovers a bigger portion of the key.

Introduction

A *periodic polyalphabetic cipher* with period *d* encrypts a *plaintext* message into *ciphertext* by replacing each plaintext letter with a letter from one of *d* substitution alphabets. The *d* alphabets are typically selected from a sequence of related alphabets that is repeated as often as necessary to encrypt the message.

In polyalphabetic encryption a message is broken into *d* columns, each encrypted by a different alphabet. The longer the period, *d*—the more alphabets used–the more difficult it is to unscramble the message.

Ciphertext only cryptanalysis is the process of recovering the plaintext directly from the ciphertext message without access to the message key. The search space for this type of problem is much too large for brute force approaches. For a polyalphabetic cipher with a 26-letter alphabet and period *d* the key space contains $26! \times 26^{d-1}$ possible keys. That gives approximately 10^{32} keys for a period of length 5 and 10^{53} keys for a period of length 20.

Several studies have used the genetic algorithm (GA) successfully in ciphertext only cryptanalysis (Clark and Dawson 1997, Matthews 1993, Morelli, Walde and Servos 2004, Morelli and Walde 2003, Spillman 1993 and Spillman, Janssen and Nelson 1993). Delman (2004) provides a recent study that compares various efforts to apply GAs cryptography. GA approaches evolve the message key(s) by repeatedly decrypting the message and measuring how close the resulting text is to plaintext.

For polyalphabetic ciphers, the search is made more difficult by the fact that each letter in two-letter (*bigram*), three-letter (*trigram*), or four-letter (*tetragram*) sequences is encrypted from a different alphabet. In this study, we show that a collection of GAs working in parallel can be used to recover increasingly longer portions of the key. As described more fully in the discussion, our results compare favorably to other approaches reported in the literature.

Polyalphabetic Ciphers

Following (King 1994), a polyalphabetic cipher can be defined as follows. Given *d* cipher alphabets $C_1 \ldots C_d$, let $g_i : A \rightarrow C_i$ be a mapping from the plaintext alphabet *A* to the i^{th} cipher alphabet $C_i (1 \leq i \leq d)$. A plaintext message $M = m_1 \ldots m_d m_{d+1} \ldots m_{2d} \ldots$ is enciphered by repeating the sequence of mappings $g_1, \ldots, g_d$ every *d* characters, giving

$$E(M) = g_1(m_1) \ldots g_d(m_d) g_1(m_{d+1}) \ldots g_d(m_{2d}) \ldots \quad (1)$$

Although in the most general case the cipher alphabets are completely unrelated, in many classical polyalphabetic ciphers the alternative alphabets $C_2, ..., C_d$ are *shifted* versions of the mixed primary cipher alphabet, C_1. Thus, in addition to the *mixed primary alphabet*, C_1, a second key, the *shift keyword*, is used to specify both the period and the shifts used to generate the other cipher alphabets, with 'a' representing a shift of 0, 'b' a shift of 1, and so on.[1]

In shifted polyalphabetic ciphers, the alternative alphabets are derived by performing a shift modulo 26 on each letter of the mixed primary alphabet. In the following example, the second alphabet is derived form the first by shifting each of its letters by one modulo 26:

```
z e r o s u m g a b c d f h i j k l n p q t v w x y
a f s p t v n h b c d e g i j k l m o q r u w x y z
```

In this case the primary alphabet is a simple substitution alphabet generated from the primary keyword *zerosumgame*.

Thus, a polyalphabetic cipher with shifted alphabets encryption can be represented as the composition, or product, of two operations: a substitution step, to create the mixed primary alphabet, and a shift step, to derive a secondary alphabets. Substitution involves replacing a plaintext letter

[1] This type of cipher, which is sometimes called a *double-key* cipher, was first introduced in the 15th century by Alberti and extended with contributions by Trithemius, Belaso, Porta, and Vigenere. Such ciphers have been well studied, both from an historical perspective (Kahn 1967, Bauer 1997) and as the basis behind secure, contemporary ciphers (Rubin 1995).

with the corresponding letter from the mixed primary alphabet. The shift step involves shifting a letter by an arbitrary amount (modulo 26).

The order of these operations is significant and leads to two different ciphers known as Type I and Type II (Gaines 1939). In Type I, where substitution is performed first, the encryption and decryption operations can be represented as:

$$c_j = g_i(m_j) = (C_1(m_j) + k_i)(mod 26) \quad (2)$$

$$m_j = g_i^{-1}(c_j) = C_1^{-1}((c_j + 26 - k_i)(mod 26)) \quad (3)$$

where $C_1(m_j)$ represents substitution of the *j*th plaintext character, m_j, from the primary cipher alphabet C_1 and k_i represents the shift of the resulting ciphertext letter. In decryption the inverse operations are applied.

In Type II the shift step is performed before substitution, leading to the following encryption and decryption operations:

$$c_j = g_i(m_j) = C_1((m_j + k_i)(mod 26)) \quad (4)$$

$$m_j = g_i^{-1}(c_j) = (C_1^{-1}(c_j) + 26 - k_i)(mod 26) \quad (5)$$

A familiar way to represent a set of shifted alphabets is to use a *Vigenere tableau*, in which the mixed primary alphabet, C_1, is placed in the first row and alternative alphabets are placed in successive rows. The following table corresponds to a Type I cipher:

```
                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
  A: z e r o s u m g a b c d f h i j k l n p q t v w x y
K B: a f s p t v n h b c d e g i j k l m o q r u w x y z
E C: b g t q u w o i c d e f h j k l m n p r s v x y z a
Y .                Ciphertext
  Z: y d q n r t l f z a b c e g h i j k m o p s u v w x
```

The rows of the table represent the shifted alphabets. For example, the shift keyword *SYMBOL* would select the following alphabets from the above table:

```
                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
  S: r w j g k m e y s t u v x z a b c d f h i l n o p q
  Y: x c p m q s k e y z a b d f g h i j l n o r t u v w
K M: l q d a e g y s m n o p r t u v w x z b c f h i j k
E B: a f s p t v n h b c d e g i j k l m o q r u w x y z
Y O: n s f c g i a u o p q r t v w x y z b d e h j k l m
  L: k p c z d f x r l m n o q s t u v w y a b e g h i j
```

To encrypt the word "the" using the above table, we replace 't' with its corresponding letter from the *S* alphabet. This is equivalent to replacing 't' by 'p' from the mixed primary alphabet and then shifting 'p' by *s* places (modulo 26). In either case, 't' is encrypted as 'h'. The word "the" would be encrypted as "hee."

The table corresponding to a Type II cipher would be represented as follows:

```
                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
  A: z e r o s u m g a b c d f h i j k l n p q t v w x y
K B: e r o s u m g a b c d f h i j k l n p q t v w x y z
E C: r o s u m g a b c d f h i j k l n p q t v w x y z e
Y .
  S: n p q t v w x y z e r o s u m g a b c d f h i j k l
  Y: x y z e r o s u m g a b c d f h i j k l n p q t v w
  M: f h i j k l n p q t v w x y z e r o s u m g a b c d
  .
  Z: y z e r o s u m g a b c d f h i j k l n p q t v w x
```

To encrypt "the" using the shift keyword SYMBOL, we would select the corresponding letters from the *S*, *Y*, and *M* rows, giving "duk". This is equivalent to taking a letter, say 't', shifting it by, say *s*, giving 'l', and replacing the result with the corresponding letter from the C_1 alphabet, giving 'd'.

Index of Coincidence

Type II ciphers are much more difficult to analyze than Type I. To show this it is necessary to describe the *Index of Coincidence*. The IC is a measure used to distinguish text that has been encrypted with a polyalphabetic cipher from plaintext or from text that was encrypted with a single alphabet. First described by (Friedman 1920), the IC is the probability that two symbols chosen at random from a text will be equivalent. For a text of length *N* written in the standard alphabet, $A \ldots Z$, where f represents character frequencies, the IC is expressed as:

$$IC = \frac{\sum_{i=A,Z} f_i(f_i - 1)}{N(N-1).} \quad (6)$$

As Friedman showed, the IC value for English plaintext and for a simple substitution cipher will be around 0.066. Text encrypted by a polyalphabetic substitution cipher would have an IC value less than 0.066 depending on the number of alphabets used. Polyalphabetics with periods greater than 10 would have IC values that approach 0.038.

Finding the Period

Regardless of whether Type I or Type II encryption was used in creating a ciphertext, the IC can be used to find the cipher's period by performing an exhaustive search on each possible period from 1 to some maximum length. For each possible period, d_i, the plaintext is broken into d_i columns and the IC is computed for each column. If d_i is the correct period, the average IC of the d_i columns will be close to 0.066. If none of the values get close enough to 0.066, we take the d_i associated with the highest IC value. While this algorithm is not guaranteed to find the correct period, we found it to be successful in nearly 100 percent of the cases.

Type I versus Type II Cryptanalysis

Once the cipher's period is known, cryptanalysis of a Type I cipher is very straightforward. Consider the following depiction of Type I encryption (substitution then shift):

$$P_{0.066} \Rightarrow C^1_{0.066} \rightarrow C^2_{0.038} \quad (7)$$

When plaintext *P*, which has an IC of 0.066, is replaced by substitution ($\Rightarrow$) from the mixed primary alphabet, the resulting ciphertext, C^1, will have an IC value approximately

equal to plaintext ($IC \approx 0.066$). When the ciphertext is then shifted ($\rightarrow$), resulting in a new ciphertext, C^2, its IC value will be characteristic of a polyalphabetic cipher ($IC \approx 0.038$).

Given this characterization of a Type I cipher, it is easy to see that the following strategy can be used to break Type I messages:

$$P_{0.066} \Leftarrow C^1_{0.066} \leftarrow C^2_{0.038} \tag{8}$$

In other words, when the correct reverse shifts ($\leftarrow$) are applied to the ciphertext, the result will be a ciphertext with an IC value characteristic of a simple substitution cipher. The IC can be used on the intermediate cipher text, C^1, to determine when the correct reverse shifts have been applied. Thus, the shift keyword for a Type I cipher can be recovered by trying all possible reverse shifts. The correct set of reverse shifts will produce a simple substitution ciphertext, C^1 whose IC value will be close to 0.066. Analysis of the recovered simple substitution ciphertext, C^1, will lead to recovery of the mixed primary alphabet, which can then be applied ($\Leftarrow$) to recover the plaintext.

By contrast, the model for Type II encryption is as follows:

$$P_{0.066} \rightarrow C^3_{0.038} \Rightarrow C^4_{0.066} \tag{9}$$

In this case, the shifts are applied first ($\rightarrow$), resulting in ciphertext that is polyalphabetic ($IC \approx 0.038$). The shifted text is then replaced using the mixed primary alphabet ($\Rightarrow$), leading to ciphertext that is now the result of the product of two operations.

Viewed thusly, it is clear that one way to break Type II would be first to recover and apply the primary key ($\Leftarrow$) and use it to recover and apply the shift keyword ($\leftarrow$):

$$P_{0.066} \leftarrow C^3_{0.038} \Leftarrow C^4_{0.038} \tag{10}$$

However, this approach will not work because there is no effective way to determine the mixed primary alphabet solely by analyzing the intermediate ciphertext, C^3. Neither the IC nor any other statistical measure provides a means to identify the correct primary key independently of the shift keyword.

Therefore, to cryptanalyze Type II ciphers we need a strategy that composes both the substitution and shift steps in reverse order, so that the IC of the resulting text will approach that of plaintext (0.066).

A Multi-Phase GA Approach

Our approach for breaking Type II ciphers uses a multi-phase, genetic algorithm (GA) as the basic search mechanism. Once the correct period, *d*, is found, the ciphertext is divided into *d* columns. The letters in a given column have all been encrypted from the same alphabet. Moreover, the alphabets used to encrypt adjacent columns are related to each other by means of some shift, *a* to *z*.

Thus, once the correct period is found, we perform GA search on two, then three, then four adjacent columns of ciphertext. At each phase we generate a partial decryption of the columns by composing the shift and substitution steps ($\leftarrow + \Leftarrow$). This leads to the recovery of increasingly larger portions of both the shift keyword and the primary (substitution) alphabet. After completion of the four-column phase, the GA has recovered enough of the primary substitution alphabet to allow full recovery of the message. Thus, our approach goes through the following phases:

0 Determine the period, *d*, of the ciphertext.

1 For columns 1-2 of the ciphertext, use GA search to find the best 2-letter shift keyword and primary alphabet compositions.

2 Given results from phase 1, use GA search to find the best 3-letter shift keyword and primary alphabet compositions for columns 1-3 .

3 Given results from phase 2, use GA search to find the best 4-letter shift keyword and primary alphabet compositions for columns 1-4.

4 Given the partially correct primary alphabet from the previous step, find the best candidate shift keywords of length *d*.

5 Given the best candidate shift keywords, use GA search to find the best shift keyword/primary alphabet composition.

After phase 5, the ciphertext has been reduced to a simple substitution cipher, which can easily be solved.

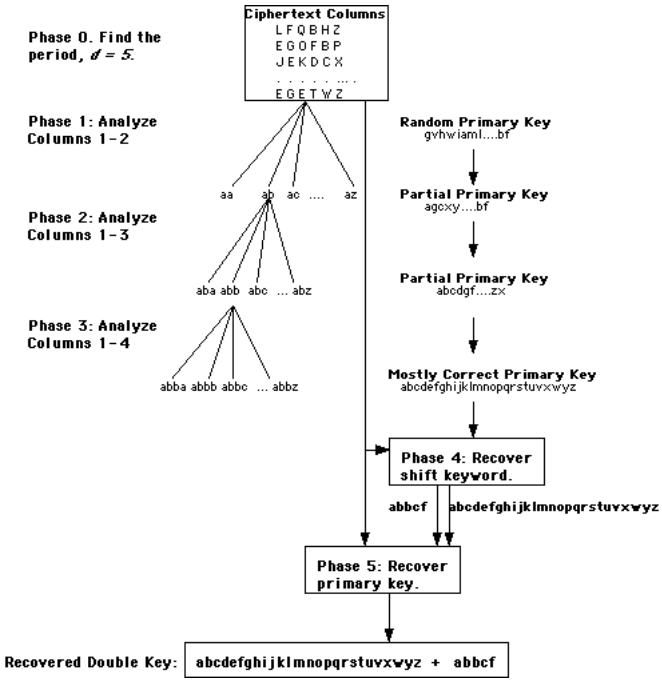


Figure 1: Multi-phase GA search on a Type II cipher with the primary alphabet *abc...z* and shift keyword *abbcf*.

Figure 1 provides a more detailed depiction of the search strategy. For two columns of ciphertext, we take all possible

2-letter shift keywords that begin with 'a': $aa, ab, \ldots, az$, and perform GA search for the best primary alphabet for each 2-letter shift.[2] These 26 searches can be performed in parallel as a competition.[3] Each of the 26 searches yields a 2-letter shift keyword and an associated primary alphabet. Each of these 26 2-letter shift keywords, $a\delta_2$, and their associated primary alphabets are used to initialize 26 GAs with all possible 3-letter keywords, $a\delta_2 a, a\delta_2 b, \ldots, a\delta_2 z$. These 26^2 searches can be performed in parallel as a competition. Each of the 26^2 searches produces an optimal 3-letter shift keyword, $a\delta_2\delta_3$, and an associated primary alphabet. These then are used to initialize 26^3 GA searches for the optimal 4-letter shift keyword and associated primary alphabet. After the 4-letter phase enough of the primary alphabet is recovered to allow the ciphertext to be broken in subsequent phases.

Fitness Measure

For the two-column phase, a fitness measure based on bigram frequencies is used to find the optimal composition of 2-letter shifts and primary alphabet. The three-column phase uses trigram frequencies, and the four-column phase uses tetragram frequencies. The tetragram fitness measure is defined as follows:

$$F_{key} = \sum_{ijkl \in C} \frac{1}{T^{tet}_{(ijkl)}} \,. \tag{12}$$

where *ijkl* represents a tetragram from the ciphertext, *C*, and T^{tet} is a table of relative tetragram frequencies for the language in which the ciphertext is written. (Our language statistics are derived from the book *Tom Sawyer*, but could be based on any relevant text.) To compute the fitness of the key, we compute the sum of the reciprocals of the corresponding known frequencies for each of the tetragrams contained in the decrypted message. The lower the value of F_{key} the closer the decryption is to English plaintext. As we have shown elsewhere (Morelli, Walde and Servos 2004), this fitness function is similar to the familiar Chi-square test, but it is faster to compute.

GA Search Details

All GA searches use populations of 208 (26×8) individuals. By using a multiple of 26 we get eight different primary alphabets for each of the 26 partial shift keys. This appears to give enough initial variation to allow the searches to succeed. We use an *elitist* strategy for selection, meaning we keep only the fittest individuals, as well as for mutation, meaning we keep only mutants that improve the pool. The parameters that control the crossover and mutation rates are set to 0.9 and 0.5 respectively, which means that proposed crosses between two individuals occur approximately 90% of the time and that approximately half of the resulting new individuals are mutated. Because our GA appears to give reasonably good performance, we have not experimented with these settings.

Each individual contains a representation of the primary alphabet and the shift keyword. Initially, in phase 1, individuals are given a random primary alphabet and one of the 26 possible 2-letter shift keywords. The initial primary alphabets are randomly generated permutations of $a \ldots z$. During each generation of GA search, pairs of selected individuals are mated by exchanging random portions of their respective primary alphabets in such a way that a valid permutation of $a \ldots z$ is maintained in the offspring. Mutation of certain offspring is performed by swapping two letters at random in their primary alphabets. After the genetic operations are performed, the keys are used to decrypt a two-, three-, or four-column portion of the message and the fitness of each individual is measured. The fittest individuals are selected to survive into the next generation.

Implementation and Testing

A fully parallel implementation of the algorithm described in the preceding section would require 26^3 processors. Phase 1 uses only 26 of the processors and phase 2 uses only 26^2 processors. During each phase, each processor would run a GA search for the optimal combination of primary alphabet and partial shift keyword. If all branches of the tree in Figure 1 are explored, then after phase 4, the optimal candidate would clearly emerge. It will have the first four letters correct in its shift keyword, and most of the mixed primary alphabet will be recovered. This enables the algorithm to find the best candidates for the full shift keyword in phase 5 and, given these, the complete primary alphabet can be recovered.

Success with this algorithm can actually be achieved with many fewer than 26^3 processors. At phase 2 the correct 2-letter keyword and associated primary alphabet will typically be in the top half of the distribution of top 2-letter GAs. In phase 3, the correct candidate will typically be among the top 5 candidates. In phase 4, the correct candidate is frequently among the top two or three candidates.

In order to determine the amount of parallelism required to guarantee the success of the search, we kept track of where the correct shift keyword occurred among the 26 re-

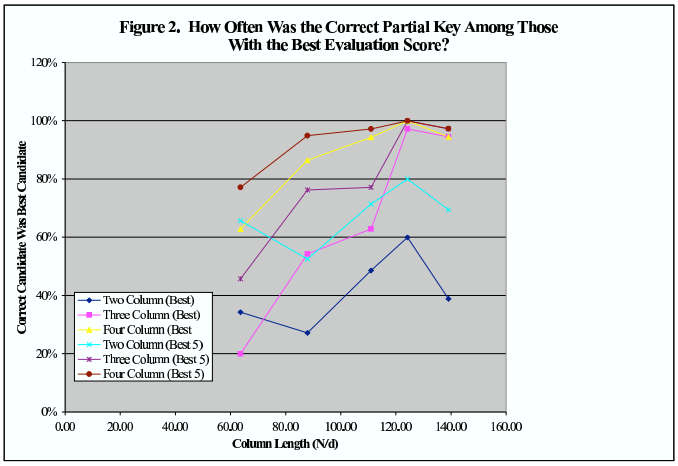
[2] It is a well known characteristic of polyalphabetic ciphers that neither the primary alphabet nor the shift keyword are unique. Instead they belong to a set of 26 pairs of keys, any pair of which leads to the same encryption (Sinkov 1966). For any actual keyword and primary alphabet, it is possible to reduce the shift keyword to one beginning with 'a' provided a similar reduction is made to the primary alphabet. For example, the actual shift keyword is "symbol" and the corresponding primary alphabet is A_i, then the same ciphertext would be produced by the shift keyword "tzncpm" associated with the alphabet A_j, where A_j is derived from A_i by shifting each letter by one, just as each letter in their respective shift keywords is shifted by one. This relationship can be expressed algebraically as follows:

$$C = S(A(P)) = S(s_k(s_{26-k}(A(P)))). \tag{11}$$

where *C* and *P* are the ciphertext and plaintext respectively, *S* is some periodic shift, *A* is the primary substitution alphabet, and s_k is a shift of *k* letters.

[3] The use of parallel search here would not affect the outcome of the search but merely its speed.

sults at each phase. These results are summarized in Figure 2 for messages encrypted with period $d = 26$. With around 100 characters per column, the correct 2-letter shift keyword was the best value found approximately 40% of the time. The correct 3-letter shift keyword was the best value found approximately 60% of the time, and correct 4-letter shift was the best value approximately 90% of the time. For these same messages, the correct shifts were among the top-5 best candidates identified by the GA approximately 60%, 80%, and 95% of the times, respectively. Note that with 120 characters per column, the correct candidate in the 3-letter and 4-letter searches were among the top-5 best candidates in close to 100% of the trials.



The data in Figure 2 provide some basis on which to estimate the actual number of processors that would be needed for a practical implementation of the parallel algorithm. With 26×5 processors, the correct partial shift keyword and primary alphabet would emerge from phase 4 at least 70% of the time for messages with a column length of at least 100 characters.

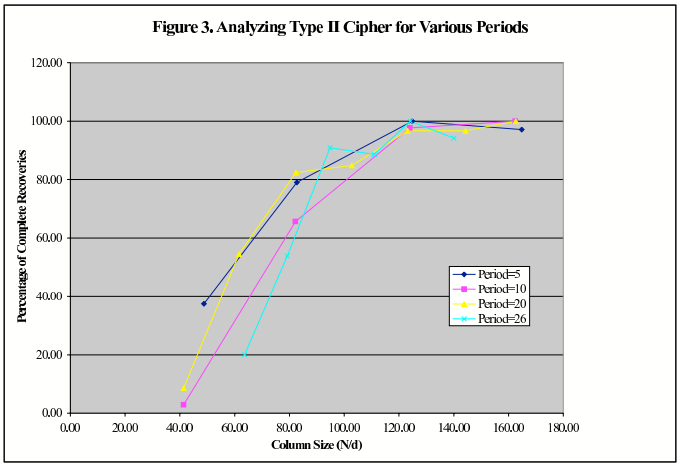
Simulation Strategy

Because we did not have access to parallel processing resources, we ran a sequential simulation of the algorithm on a single processor. For phases 1-3, the simulation conducted 26 GA searches and selected the *correct* 2-, 3-, or 4-letter shiftword prefix and its associated primary alphabet to pass on to the next phase. This allowed us to guarantee that the correct shift keyword prefix was being passed on to the next phase. Of course, knowledge about the correct candidate was not used in the actual GA searches.

Experimental Results

The sequential simulation was tested on a total of 125 messages, selected at random from the book *Oliver Twist*, ranging in length from approximately 250 to 4000 characters, with periods of length 5, 10, 20, and 26. Longer messages (3000-4000 characters) were used with the longer periods. Shorter messages (250-500 characters) were used with shorter periods. More than 750 test runs were made, approximately six runs per message.

Figure 3 summarizes the results. The horizontal axis represents the average column length of the encrypted messages—i.e., message length divided by period. The vertical axis represents the percentage of times that the message was completely decrypted. A score of 100% means that the both the shift keyword and the primary alphabet were completely recovered in every case. A 50% success rate means the algorithm was able to completely recover both keys in only half the cases. The lines in the graph represent ciphers with different periods (5, 10, 20, or 26).



As expected, the results show clearly that success of the algorithm depends on the column length. Regardless of the period, good success rates were achieved starting at approximately 100 characters per column. Close to 100% success was achieved starting at around 120 characters per column.

The success rate was much lower for messages with 60 or fewer characters per column. Again, however, success rate means the percentage of times that a message was completely decrypted. Given the stochastic nature of GAs, it would be necessary to perform multiple runs of the search in order to break shorter messages. In our experiments, there were very few messages, even those with relatively short column lengths, that were *never* successfully decrypted, although a message with a column length of 60 characters might be successfully decrypted only once or twice in six attempts. We return to this point in the discussion.

Discussion

In terms of computational effort, the sequential simulation described in the preceding section examined an average of 2.5 million keys per message. Of course, because the simulation uses the correct shifts at each phase, this is a very optimistic lower bound on search cost. A fully-parallelized implementation would utilize approximately 26 times more computation during phase 2 and 26^2 more computation during phase 3 of the search, which would lead to examination of around 400 million keys per message. This would be the upper bound on search cost. As we described above, a practical parallel search would require many fewer processors and would be able to break messages by examining far fewer keys.

Although 400 million keys seems like a large search, it represents a miniscule portion of the key space. For the full parallel algorithm, the proportion of the key space examined is less than 10^{-23} for a shift key of length 5 and is less than 10^{-44} for a shift key of length 20. This compares favorably to the proportion of approximately 10^{-21} for GA solutions to simple substitution ciphers as described in an earlier study (Morelli, Walde and Servos 2004). Notice that for longer shift keys our algorithm requires a proportionally longer ciphertext to be successful but only the very efficient step 4 of the algorithm requires additional computation.

Our experimental results compare favorably with other ciphertext-only analyses of polyalphabetic ciphers. In (Carroll and Robbins 1987) a Bayesian attack was used on a general (not shifted) polyalphabetic cipher with a period of 3, but 100% success was not achieved. In (Carroll and Robbins 1989) limited success was achieved in the analysis of product ciphers consisting of the composition of substitution and transposition steps. As noted there, the order in which the operations are performed determined the difficulty of the analysis.

In (Clark and Dawson 1997) a GA was used to successfully analyze a Vigenere cipher with a period of 3. A Vigenere cipher is much easier to break than a Type I cipher because its primary alphabet is not mixed. King (1994) used a probabilistic relaxation technique to successfully attack both Vigenere and Type I ciphers. For Vigenere ciphers, 100% decryption was attained for periods up to 30 at around 125-150 characters per column. For Type I ciphers, probabilistic relaxation was successful for periods up to 7 at around 125-150 letters per column. Our approach gives comparable results for the much more difficult Type II ciphers. Because King's approach does not appear to make any assumptions about the order of the shift and substitute operations in calculating the probabilistic alphabets, it could likely be applied to Type II ciphers. However, we would expect that the results of that approach would be no better than our Type II results.

It is clear that what makes Type II ciphers difficult to break is the lack of a statistical measure or technique that would allow the cryptanalyst to reverse the shifts. Because the shifts are applied first, the monoalphabetic substitution step is being applied to a polyalphabetic cipher. Therefore the technique must be capable of de-composing, if you will, the product of two operations. One technique that might apply here is the *expectation-maximization algorithm*, a maximum-likelihood estimation technique that has been used successfully on problems such as separating signal from noise (Moon 1996).

Plans for the Future

A Type II cipher with a period of 26 is equivalent to a single rotor in a rotor machine that uses a random rotation scheme (as opposed to an odometer-like scheme). We are currently exploring whether our approach can be applied to the analysis of a rotor machine with more than one rotor.

References

1. F. L. Bauer. *Decrypted Secrets: Methods and Maxims of Cryptology*. Springer-Verlag, Berlin, 1997.
2. J. Carroll, L. Robbins. The automated cryptanalysis of polyalphabetic ciphers. *Cryptologia*, 11(4): 193-205, 1987.
3. J. Carroll, L. Robbins. Computer cryptanalysis of product ciphers. *Cryptologia*, 13(4): 303-326, 1989.
4. A. Clark, E. Dawson. A parallel genetic algorithm for cryptanalysis of the polyalphabetic substitution cipher. *Cryptologia*, 21(2): 129-138, 1997.
5. B. Delman. Genetic algorithms in cryptography. *Master's Thesis*, RIT: http://hdl.handle.net/1850/26, July 2004.
6. W. F. Friedman. *The Index of Coincidence and Its Application in Cryptography. Riverbank Publication No. 22*. Riverbank Labs, Geneva IL, 1920.
7. H. F. Gaines. *Cryptanalysis: A Study of Ciphers and Their Solution*. Dover Publications Inc., New York, 1939.
8. D. Kahn. *The Codebreakers*. Macmillan Co., New York, 1967.
9. J. C. King. An algorithm for the complete automated cryptanalysis of periodic polyalphabetic substitution ciphers. *Cryptologia*, 18(4):332-355, 1994.
10. R. Matthews. The use of genetic algorithms in cryptanalysis. *Cryptologia*, 17(2):187-201, 1993.
11. T. K. Moon. The expectation-maximization algorithm. *IEEE Signal Processing Magazine*, Nov. 1996.
12. R. Morelli, R. Walde, W. Servos. A study of heuristic approaches for breaking short cryptograms. *International Journal on Artificial Intelligence Tools*, 13(1):45-64, 2004.
13. R. Morelli, R. Walde. A word-based genetic algorithm for cryptanalysis of short cryptograms. *Proceedings of the 2003 Florida AI Rsch. Symp.* FL AI Research Soc., 229-233, 2003.
14. F. Rubin. Designing a high security cipher. http://www.contestcen.com/crypt003.htm, 1995.
15. A. Sinkov. *Elementary Cryptanalysis*. Yale University Press, New Haven, CT, 1966.
16. R. Spillman. Cryptanalysis of knapsack ciphers using genetic algorithms. *Cryptologia*, 17(4):367-377, 1993.
17. R. M. Spillman, B. Janssen, B. Nelson. Use of a genetic algorithm in the cryptanalysis of simple substitution ciphers. *Cryptologia*, 17(1):31-44, 1993.