Cryptologia
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/ucry20

A PARALLEL GENETIC ALGORITHM FOR CRYPTANALYSIS OF THE POLYALPHABETIC SUBSTITUTION CIPHER

Andrew Clark [a] & Ed Dawson [a]
[a] Information Security Research Centre, Queensland University of Technology, GPO Box 2434, Brisbane, 4001, Queensland, AUSTRALIA. Email: {aclark,dawson}@fit.qut.edu.au.
Published online: 04 Jun 2010.

A PARALLEL GENETIC ALGORITHM FOR CRYPTANALYSIS OF THE POLYALPHABETIC SUBSTITUTION CIPHER

Andrew Clark and Ed Dawson

ADDRESS: Information Security Research Centre, Queensland University of Technology, GPO Box 2434, Brisbane, 4001, Queensland, AUSTRALIA. Email: {aclark,dawson}@fit.qut.edu.au.

ABSTRACT: A new method for attacking the simple substitution cipher is presented which utilises a parallel version of the genetic algorithm. A suitable strategy is devised which allows communication between a number of parallel nodes each solving a separate part of the problem. An analysis of the fitness function is also performed.

KEYWORDS: Parallel genetic algorithm, automated cryptanalysis, substitution cipher.

1 INTRODUCTION

Cryptanalysts are continually seeking new methods for automating attacks on ciphers. Traditionally, the new technique is applied to a simple cipher in the first instance, following which it is scrutinized for possible use against more complex cryptosystems. In this paper, we present a new strategy for attacks on the polyalphabetic substitution cipher. The attack is realised using a parallel genetic algorithm. Serial genetic algorithms have been shown to be successful in cryptanalysing both transposition ciphers (see [6]) and simple substitution ciphers (see [8]). The work presented here is essentially an extension of the paper by Spillman et. al. in [8] to a polyalphabetic substitution cipher.

The genetic algorithm is well suited to parallel implementations, a fact clearly illustrated by the abundance of related research papers (for example, see [1, 3, 5, 6]). The potential of such strategies is still widely unexplored in the area of cryptanalysis. The polyalphabetic substitution cipher is suitable for parallel attacks since it can be solved as a number of simple (or monoalphabetic) substitution ciphers simultaneously.

The genetic algorithm and the parallel genetic algorithm are discussed in some detail in Section 2. A detailed description of the implementation is provided in Section 3. The design phase is also discussed in this section. Section 4 provides a summary of the results obtained for various experiments on the parallel genetic algorithm. An analysis of the fitness function proposed in this paper is also given. Finally, in Section 5, some conclusions are made regarding the effectiveness of the attack that has been presented.

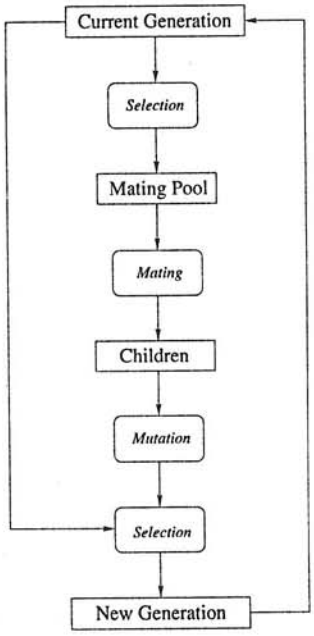

Figure 1. Typical GA Structure

2 THE GENETIC ALGORITHM

The genetic algorithm is widely discussed in the literature and numerous examples of its applications can be found. In this section a brief description of the basic genetic algorithm is given, followed by a more detailed discussion of the possibilities for the genetic algorithm when implemented in parallel. The choice of a particular parallel genetic algorithm can be influenced by the structure of the problem being solved or the availability of certain computer hardware.

As its name suggests, the genetic algorithm uses concepts from the field of genetics. The basic data object in the genetic algorithm is a pool of feasible

solutions to the problem being solved. In general, the larger the solution pool, the greater the "diversity" of solutions that are trialled during the search process. From the solution pool a number of parents are chosen for mating to produce offspring (new solutions). These offspring are mutated in some manner and then combined with the previous generation to give a new generation. The mating and mutation operators are specific to the solution representation which is usually (but not necessarily) related to the problem being solved. Each solution has associated with it a fitness which is used to rank a solution against its peers. Fitness usually comes into play in the mating and mutation operators as well. The overall effect of the genetic algorithm is that the fittest solutions remain, while the less fit ones are discarded as time progresses. The structure of a typical (serial) genetic algorithm is given in Figure 1.

The adaptation of a serial genetic algorithm to a parallel one is largely a problem-specific task. Also, as alluded to above, there is a requirement for certain software/hardware when implementing a parallel design (at least this is a requirement in order to gain maximum benefit from such a design). Two possible strategies are presented here.

Strategy I: Parallelise each of the operations within a single genetic algorithm. That is, parallelise the selection, mating and mutation phases of the algorithm. For example, consider a problem where a gene pool of size N is maintained. Now assume there exists N separate processing nodes capable of performing one (or all) of the phases mentioned above. Then, instead of processing each of the N solutions in the pool serially, all solutions could be processed simultaneously (in parallel), providing a speed-up of order N^3 (N times for each of the three phases). Figure 2 illustrates such a scheme diagrammatically. An example of an application which uses this strategy is given in [5].

Strategy II: Have a number of basic genetic algorithms solving different parts of the same problem in parallel. This strategy is used in the following section to perform an attack on the polyalphabetic substitution cipher. The paper by Fukuyama and Chiang (see [4]) illustrates an application of a similar strategy to this one.

3 IMPLEMENTATION SPECIFICS

In this section the details of the parallel genetic algorithm, as implemented, are given. First, though, it is important to understand the problem being solved. The

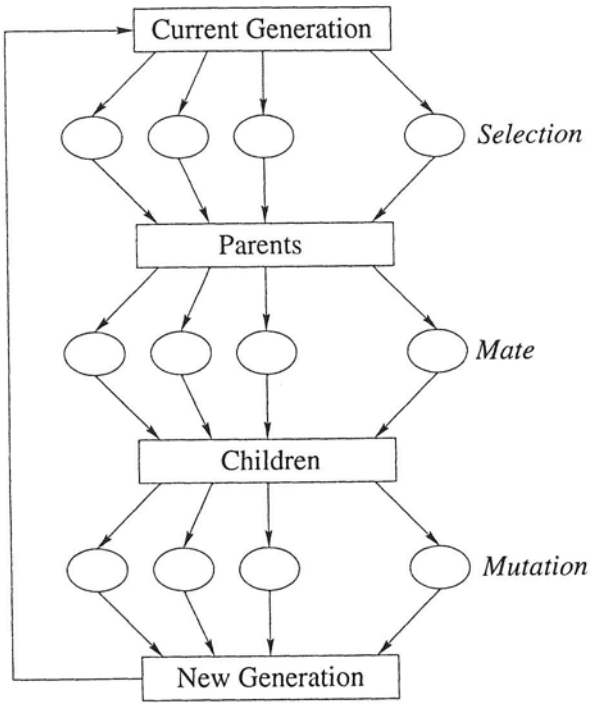

Figure 2. A Possible Parallel Heuristic for the GA.

polyalphabetic substitution cipher works on blocks of characters of length, say M, where M is known as the period of the cipher. A simple substitution cipher exchanges every character in the message with some other character according to a key where the size of the key is the same as the size of the message alphabet. A polyalphabetic substitution cipher has M such keys and each element of the block is enciphered using a different key. Thus, every M^{th} ciphertext is encrypted using the same key. In the attack presented in this paper M is assumed known. It is usually a simple task to find M using one of the number of methods documented in the literature (e.g., the Kasiski Method [2]).

Attacks on the substitution cipher traditionally make use of the underlying language statistics which remain unchanged upon encryption. Given sufficient intercepted ciphertext it is almost always possible to recover a significant amount of the original message. The language statistics are used in the calculation of the fitness of a given key. In general n-gram statistics are used, where an n-gram is a block of n characters. For example, `TH` is a very common bigram in the English language, and `E` is often the most frequent unigram. The fitness used here compares the statistics of the message decrypted with the proposed key (K) with the known language statistics as in Equation 1,

$$F(K) = w_1 \sum_{i=1}^{N} \left(L_i^{(1)} - D_i^{(1)}\right)^2 + w_2 \sum_{i,j=1}^{N} \left(L_{i,j}^{(2)} - D_{i,j}^{(2)}\right)^2 + w_3 \sum_{i,j,k=1}^{N} \left(L_{i,j,k}^{(3)} - D_{i,j,k}^{(3)}\right)^2 \quad (1)$$

where $L^{(1)}$, $L^{(2)}$ and $L^{(3)}$ are the known language unigram, bigram and trigram statistics, and $D^{(1)}$, $D^{(2)}$ and $D^{(3)}$ are the unigram, bigram and trigram statistics of the message decrypted with key K. It should be noted that the two neighbouring keys are required in order to obtain bigram and trigram statistics, however the fitness of only one key is being assessed using (1). The weights w_1, w_2 and w_3 can be varied to allow more or less emphasis on particular statistics. Some experimental results which indicate an optimal choice of the weights is given in Section 4.

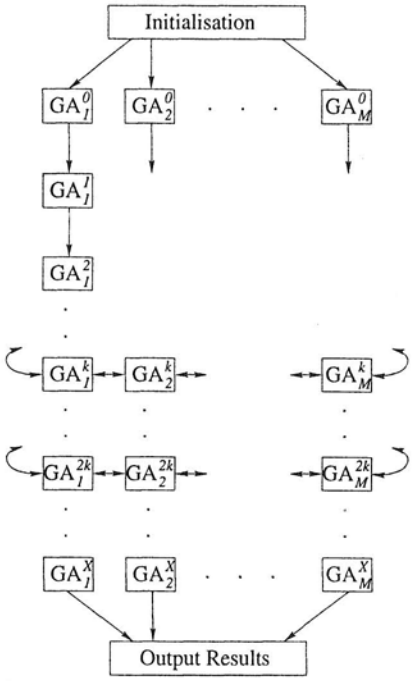

Figure 3. Parallel GA Structure for Attacking Polyalphabetic Substitution Cipher

The strategy used here to solve the polyalphabetic substitution cipher involves M separate genetic algorithms attempting to find one of the M simple substitution cipher keys simultaneously. Each genetic algorithm can be run on a separate node of a parallel computer. Hence, the attack is essentially equivalent

in complexity to an attack on a simple substitution cipher. There is, however, an additional requirement that each node communicate with its two neighbours in order to exchange best keys for the calculation of bigram and trigram statistics. This communication need not take place in every iteration of the genetic algorithm but should be frequent enough to ensure that the fitness calculation is as accurate as possible.

Figure 3 illustrates the strategy used here. GA_j^i denotes the i^{th} iteration of the GA trying to solve the j^{th} simple substitution cipher (in the block of M). Each GA communicates with its two neighbours every k iterations.

The mating and mutation functions were the same as those devised by Spillman, et. al., in [8]. The new generation is chosen to be the best (i.e., most fit) of the parent and child generations. Each genetic algorithm (GA_j) has the following structure:

1. *Input:* Given language statistics, ciphertext, block size (M), the position within the block being solved, j ($j \in 1, \ldots, M$) and the frequency of inter-node communications, k.

2. *Initialisation:* Generate a random pool of N keys (each key is just a permutation of the alphabet characters) and calculate the fitness of each key using unigram statistics only (i.e, $w_1 = 1, w_2 = w_3 = 0$). It is not possible to use bigram and trigram statistics since neighbouring keys are, as yet, unknown.

3. *Iterate:* $i = 1, \ldots,$ `Num_Iterations`.

   - Select $N/2$ distinct pairings from the gene pool as the parents for the new generation. The parents are chosen pseudo-randomly, with a bias toward the most fit keys in the current generation. The same parent may appear in a number of pairings.
   - Parents are combined using the mating function in [8]. This operation produces N children.
   - Each child then undergoes mutation, once again using the operators described in [8].
   - Calculate the fitness of each of the children keys.
   - The newly created gene pool and the current gene pool are combined and sorted by fitness. The best N are then chosen to become the new current generation.
   - If $i \bmod k \equiv 0$, send the most fit key to, and receive the most fit key from $GA_{(j-1) \bmod M}$ and $GA_{(j+1) \bmod M}$.

4. *Output:* Return the best key from the final iteration as the key for position j in the block.

Given sufficient iterations, and provided the message has statistics which are reasonably indicative of the language, the above algorithm will converge to a good solution.

4 EXPERIMENTAL RESULTS

In this section some results are presented. Two experiments were performed. The first was to investigate the amount of ciphertext required in order to recover a considerable amount of the message text. This experiment serves the dual purpose of displaying how effective the parallel genetic algorithm is in the attack on the polyalphabetic substitution cipher. The second experiment which results are given for, displays the optimal choice of weights for the fitness function (1).

The algorithm as described above was implemented in the C programming language on a Sun SparcStation with 4 processors. The excellent public domain software package PVM (Parallel Virtual Machine - see [7]) was used in order to write code which allowed communications between different processes. PVM can be run on a number of networked computers allowing a single task to share a relatively large number of processors. The experiments were conducted on text containing 27 characters, `A` to `Z` and the space symbol.

Throughout the experiments a number of parameters were fixed. They were:

- Gene pool size (N) 40
- Period (M) 3
- Iterations of GA (`Num_Iterations`) 200
- Mutation rate (see [8]) 0.05
- Iterations between slave-to-slave communications (k) 10

In the first experiment the weights w_1, w_2 and w_3 were set to 0.4, 0.6 and 0.0, respectively, for the first 100 iterations and 0.2, 0.3 and 0.5, respectively, for the final 100 iterations. Doing this dramatically improves the execution time of the algorithm since trigram statistics are not being calculated in the first 100 iterations. As will be seen from the second experiment, trigram statistics play a major role in determining a good solution, however, by using a fitness function which is faster to compute a time advantage is gained in the first half of the search.

The amount of message correctly decrypted and the number of key elements correctly determined was recorded for a number of experiments with amounts of

ciphertext varying from 200 to 1400 in increments of 200. The results are shown in Figure 4. Each point on the plots was determined by running the algorithm on ten different messages, and five times for each message. The best of the five runs for each message was recorded and the average from the ten messages taken.

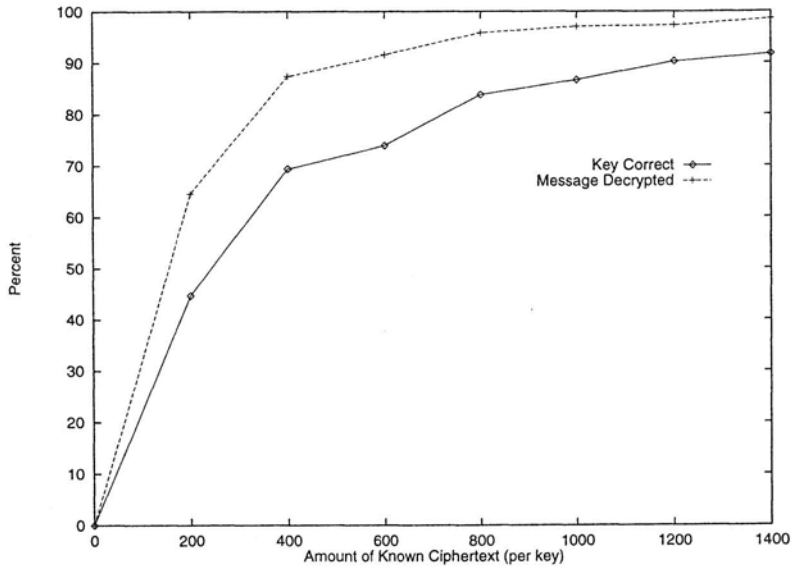

Figure 4. Amount of known ciphertext vs. percentage of key and message recovered.

It can be seen that with 600 known ciphertext characters per key (i.e., $600 \times M$ characters in all), over 90% of the message was correctly decrypted and on average more than 70% (or 19 out of 27) of the key elements were correctly determined. This result shows that, for the text used, the 8 least common characters in the message account for approximately 10% of the message.

The second experiment which was performed was designed to determine the optimal choice of weights w_1, w_2 and w_3 from the fitness function (1). Since an exhaustive trial of all combinations of weights is clearly infeasible the following restrictions were applied:

$$\begin{aligned} w_1, w_2, w_3 &\in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}, \text{ and} \\ \sum_{i=1}^{3} w_i &= 1.0. \end{aligned}$$

The are 66 combinations of weights satisfying the above restrictions. For each combination, the algorithm was run on ten different messages and twice for each message. Of the two runs for each message the best result was taken and averaged over the ten different messages. The results for each weight are

displayed in Figure 5. It can be seen that the best results were obtained when $w_3 = 0.8$, indicating that a large weighting on the trigram statistic is important. Also, a number of good results were obtained when $w_1 = 0.1$ indicating that the higher order statistics are much more powerful, which is as expected.

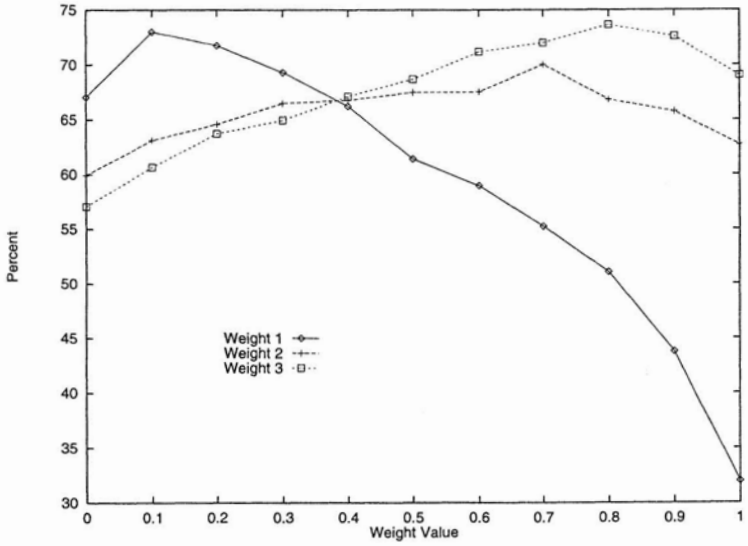

Figure 5. Weight value vs. percent of message recovered.

5 CONCLUSIONS

In conclusion, a new and efficient method for solving the polyalphabetic substitution cipher has been presented. We employ a parallel strategy where a number of processes run simultaneously and share important information. An optimal choice of weights for the fitness function has been suggested through extensive computational experiments.

Further applications of parallel strategies in the field of cryptanalysis need to be investigated. Parallel computation can lead to dramatic improvements in performance of a number of possible areas. This is especially appropriate given the ever increasing number and ever decreasing cost of parallel computing machines and the move towards a fully distributed computing environment.

ACKNOWLEDGEMENTS

The authors wish to thank course-work Masters student Harrie Nieuwland for his preliminary investigations in this work.

REFERENCES

1. Brown, Donald E., Christopher L. Huntley, and Andrew R. Spillane. 1989. A parallel genetic heuristic for the quadratic assignment problem. *International Conference on Genetic Algorithms.* 406–415.

2. Denning, Dorothy E. R. 1982. *Cryptography and Data Security.* Reading MA: Addison-Wesley.

3. Duncan, Bruce S. 1993. Parallel evolutionary programming. In D. B. Fogel and W. Atmar, editors, *Proceedings of the 2nd Annual Conference on Evolutionary Programming.* pp. 202–208. La Jolla CA. EP Society.

4. Fukuyama, Yoshikazu and Hsaio-Dong Chiang. 1996. A parallel genetic algorithm for generation expansion planning. *IEEE Transactions on Power Systems* May. 955–961.

5. Hauser,R. R. Männer, and M. Makhaniok. 1995. NERV: A parallel processor for standard genetic algorithms. In *Proceedings of the 9th International Parallel Processing Symposium.* pp. 782–289. *IEEE Comput. Soc. Press.* April.

6. Matthews, Robert A. J. 1993. The use of genetic algorithms in cryptanalysis. *Cryptologia.* 17(2): 187–201.

7. Unknown. 1996. (Email: *pvm@msr.epm.ornl.gov*). PVM: Parallel Virtual Machine. URL: `http://www.epm.orln.gov/pvm/pvm_home.html`. Accessed on May 29 1996. Last updated May 21 1996.

8. Spillman, Richard, Mark Janssen, Bob Nelson, and Martin Kepner. 1993. Use of a genetic algorithm in the cryptanalysis of simple substitution ciphers. *Cryptologia.* 17(1): 31–44.

BIOGRAPHICAL SKETCHES

Andrew Clark received his BSc from the University of Queensland in 1991. In 1992 he obtained a BAppSc (Hons) in mathematics from the Queensland University of Technology. Currently he is undertaking PhD studies at the Queensland University of Technology in the field of cryptology. His interest areas include automated cryptanalysis, network security and numerical optimisation.

Ed Dawson obtained a BSc in 1969 from the University of Washington, a MA in 1974 from the University of Sydney, a MLitStu in 1981 and a MSc in 1985 both from the University of Queensland, and a PhD from the Queensland University of Technology in 1991. He is a member of the IEEE and the International Association for Cryptologic Research, and a fellow of the Institute of Combinatorics and its Applications. He is an Associate Professor and deputy director of the Information Security Research Centre at the Queensland University of Technology with research and teaching interests in cryptology and error control coding.