Genetic Algorithms in Cryptography

by

Bethany Delman

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Engineering

Supervised by

Professor Dr. Muhammad Shaaban
Department of Computer Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, New York
July 2004

Approved By:

Dr. Muhammad Shaaban
Professor
Primary Advisor

Dr. Stanisław Radziszowski
Professor, Computer Science Department

Dr. Shanchieh Jay Yang
Assistant Professor

Thesis Release Permission Form

Rochester Institute of Technology
Kate Gleason College of Engineering
Master of Science, Computer Engineering

Title: Genetic Algorithms in Cryptography

Bethany Delman

Date

Abstract

Genetic algorithms (GAs) are a class of optimization algorithms. GAs attempt to solve problems through modeling a simplified version of genetic processes. There are many problems for which a GA approach is useful. It is, however, undetermined if cryptanalysis is such a problem.

Therefore, this work explores the use of GAs in cryptography. Both traditional cryptanalysis and GA-based methods are implemented in software. The results are then compared using the metrics of elapsed time and percentage of successful decryptions. A determination is made for each cipher under consideration as to the validity of the GA-based approaches found in the literature. In general, these GA-based approaches are typical of the field.

Of the genetic algorithm attacks found in the literature, totaling twelve, seven were re-implemented. Of these seven, only three achieved any success. The successful attacks were those on the transposition and permutation ciphers by Matthews [20], Clark [4], and Gründlingh and Van Vuuren [13], respectively. These attacks were further investigated in an attempt to improve or extend their success. Unfortunately, this attempt was unsuccessful, as was the attempt to apply the Clark [4] attack to the monoalphabetic substitution cipher and achieve the same or indeed any level of success.

Overall, the standard fitness equation genetic algorithm approach, and the scoreboard variant thereof, are not worth the extra effort involved. Traditional cryptanalysis methods are more successful, and easier to implement. While a traditional method takes more time, a faster unsuccessful attack is worthless. The failure of the genetic algorithm approach indicates that supplementary research into traditional cryptanalysis methods may be more useful and valuable than additional modification of GA-based approaches.

Contents

1 Introduction **1**

1.1 Document Overview 2

2 Genetic Algorithms **4**

3 Cryptography **9**

3.1 Monoalphabetic Substitution cipher 9

3.2 Polyalphabetic Substitution cipher 11

3.3 Permutation/Transposition ciphers 12

3.3.1 Permutation cipher 13

3.3.2 Transposition cipher 13

3.4 Knapsack ciphers 15

3.4.1 Merkle-Hellman Knapsack cipher 16

3.4.2 Chor-Rivest Knapsack cipher 18

3.5 Vernam cipher 22

3.6 Comments 22

4 Relevant Prior Works **23**

4.1 Spillman et al. - 1993 25

4.1.1 Comments 26

4.2 Matthews - 1993 27

4.2.1 Comments 30

4.3 Spillman - 1993 31

4.3.1 Comments 33

4.4 Clark - 1994 34

4.4.1 Comments 35

4.5 Lin, Kao - 1995 35

4.5.1 Comments 36

4.6 Clark, Dawson, Bergen - 1996 37

4.6.1 Comments 38

4.7 Clark, Dawson, Nieuwland - 1996 39

4.7.1 Comments 41

4.8 Clark, Dawson - 1997 41

4.9 Kolodziejczyk - 1997 42

4.9.1 Comments 42

4.10 Clark, Dawson - 1998 43

4.10.1 Comments 44

4.11 Yaseen, Sahasrabuddhe - 1999 45

4.11.1 Comments 47

4.12 Gründlingh, Van Vuuren - submitted 2002 47

4.12.1 Comments 50

4.13 Overall Comments 51

5 Traditional Cryptanalysis Methods 52

5.1 Distinguishing among the three ciphers 53

5.2 Monoalphabetic substitution cipher 53

5.3 Polyalphabetic substitution cipher 54

5.4 Permutation/Transposition ciphers 56

5.5 Comments 58

6 Attack Results 59

6.1 Methodology 59

6.2 Classical Attack Results 61

6.3 Genetic Algorithm Attack Results 64

6.4 Comparison of Results 71

7 Extensions and Additions 74

7.1 Genetic Algorithm Attack Extensions 74

7.1.1 Transposition Cipher Attacks 75

7.1.2 Permutation Cipher Attack 76

7.1.3 Comments 77

7.2 Additions 77

7.2.1 Comments 79

8 Evaluation and Conclusion 81

8.1 Summary 81

8.2 Future Work 83

8.3 Conclusion 83

Bibliography 85

List of Figures

4.1 Substitution Cipher Family and Attacking Papers 23

4.2 Permutation Cipher Family and Attacking Papers 24

4.3 Knapsack Cipher Family and Attacking Papers 24

4.4 Vernam Cipher Family and Attacking Paper 24

6.1 Classical Substitution Attack 62

6.2 Classical Vigenère Attack 63

6.3 Classical Permutation Attack 64

6.4 Classical Transposition Attack 65

6.5 Effect of Number of Generations at Population Size 10 67

6.6 Effect of Number of Generations at Population Size 20 67

6.7 Effect of Number of Generations at Population Size 40 68

6.8 Effect of Number of Generations at Population Size 50 68

6.9 Effect of Population Size at 25 Generations 69

6.10 Effect of Population Size at 50 Generations 69

6.11 Effect of Population Size at 100 Generations 70

6.12 Effect of Population Size at 200 Generations 70

6.13 Effect of Population Size at 400 Generations 71

Glossary

block	A sequence of consecutive characters encoded at one time
block length	The number of characters in a block
chromosome	The genetic material of a individual - represents the information about a possible solution to the given problem
cipher	An algorithm for performing encryption (and the reverse, decryption) - a series of well-defined steps that can be followed as a procedure. Works at the level of individual letters, or small groups of letters
ciphertext	A text in the encrypted form produced by some cryptosystem. The convention is for ciphertexts to contain no white space or punctuation
crossover (mating)	Crossover is the process by which two chromosomes combine some portion of their genetic material to produce a child or children
cryptanalysis	The analysis and deciphering of cryptographic writings or systems
cryptography	The process or skill of communicating in or deciphering secret writings or ciphers
cryptosystem	The package of all processes, formulae, and instructions for encoding and decoding messages using cryptography
decryption	Any procedure used in cryptography to convert ciphertext (encrypted data) into plaintext
digram	Sequence of two consecutive characters
encryption	The process of putting text into encoded form

fitness	The extent to which a possible solution successfully solves the given problem - usually a numerical value
generation	The average interval of time between the birth of parents and the birth of their offspring - in the genetic algorithm case, this is one iteration of the main loop of code
genetic algorithm (GA)	Search/optimization algorithm based on the mechanics of natural selection and natural genetics
key	A relatively small amount of information that is used by an algorithm to customize the transformation of plaintext into ciphertext (during encryption) or vice versa (during decryption)
key length	The size of the key - how many values comprise the key
monoalphabetic	Using one alphabet - refers to a cryptosystem where each alphabetic character is mapped to a unique alphabetic character
mutation	Simulation of transcription errors that occur in nature with a low probability - a child is randomly changed from what its parents produced in mating
one-time pad	Another name for the Vernam cipher
order-based GA	A form of GA where the chromosomes represent permutations. Special care must be taken to avoid illegal permutations
plaintext	A message before encryption or after decryption, i.e., in its usual form which anyone can read, as opposed to its encrypted form
polyalphabetic	Using many alphabets - refers to a cipher where each alphabetic character can be mapped to one of many possible alphabetic characters

population	The possible solutions (chromosomes) currently under investigation, as well as the number of solutions that can be investigated at one time, i.e., per generation
scoreboard	Method of determining the fitness of a possible solution - takes a small list of the most common digrams and trigrams and gives each chromosome a score based on how often these combinations occur
trigram	Sequence of three consecutive characters
unigram	Single character
Vigenère	The specific polyalphabetic substitution cipher used in this work

Chapter 1

Introduction

The application of a genetic algorithm (GA) to the field of cryptanalysis is rather unique. Few works exist on this topic. This nontraditional application is investigated to determine the benefits of applying a GA to a cryptanalytic problem, if any. If the GA-based approach proves successful, it could lead to faster, more automated cryptanalysis techniques. However, since this area is so different from the application areas where GAs developed, the GA-based methods will likely prove to be less successful than the traditional methods.

The primary goals of this work are to produce a performance comparison between traditional cryptanalysis methods and genetic algorithm based methods, and to determine the validity of typical GA-based methods in the field of cryptanalysis.

The focus will be on classical ciphers, including substitution, permutation, transposition, knapsack and Vernam ciphers. The principles used in these ciphers form the foundation for many of the modern cryptosystems. Also, if a GA-based approach is unsuccessful on these simple systems, it is unlikely to be worthwhile to apply a GA-based approach to more complicated systems. A thorough search of the available literature found GA-based attacks on only these ciphers. In many cases, these ciphers are some of the simplest possible versions. For example, one of the knapsack systems attacked is the Merkle-Hellman knapsack scheme. In this scheme, a superincreasing sequence of length n is used as a public key. According to Shamir [23], a typical value for n is 100. In the GA-based attacks on this cipher, n was less than 16. This is almost an order of magnitude below the typical case, and makes these attacks useful only as proof-of-concept examples for learning the use of

GAs.

In other GA-based attacks, there is a huge gap in the knowledge needed to re-implement the attack and the information provided in the paper. For example, the values of p and h are vitally necessary for an implementation of the Chor-Rivest knapsack scheme. This system utilizes a finite field, $\mathbb{F}_q$, of characteristic p, where $q = p^h$, $p \geq h$, and the discrete logarithm problem is feasible. In the GA-based attack on this cipher, these values are not provided. This information hole means that this attack could have been run using trivial parameters, or parameters known to be easy to attack. This attack can not be considered remotely valid without knowing these parameters.

Many of the GA-based attacks also lack information required for comparison to the traditional attacks. The number of generations it took for a run of the GA to get to some state is not at all useful when attempting to compare and contrast attacks. The most coherent and seemingly valid attacks were re-implemented so that a consistent, reasonable set of metrics could be collected. These metrics include elapsed time required for the attack and the success percentage of each attack.

1.1 Document Overview

This thesis is broken into a number of chapters. Each chapter covers one aspect of the thesis in detail. Chapter 2 gives a brief introduction to genetic algorithms. The general form of the selection, reproduction and mutation operators are discussed, as is the general sequence of events for a GA-based approach. Chapter 3 introduces the necessary classical ciphers in detail. The algorithm for each cipher is given, in some cases including a small example to illustrate the process of using the given cryptosystem.

The next two chapters cover the different attacks found in the literature. Chapter 4 discusses the various GA-based attacks in detail. These attacks are covered in chronological order, with up to four papers attacking the same cipher or variants thereof. The following chapter, Chapter 5, details the classical attack on those ciphers chosen for further investigation. The ciphers were selected for further investigation due to the difficulty or lack thereof of the traditional attack method, as well as the clarity and quality of the GA-based work. For example, the Vernam cipher was not selected for further investigation. This cipher is unconditionally secure, meaning that an attack is successful only when there is a protocol or implementation failure. There is no basis for attacking this cipher when no mistakes have been made.

The next chapters detail the results of the traditional and GA-based attacks, as well as extensions to the GA-based attacks. Chapter 6 covers the results of the traditional attacks, and discusses those GA-based attacks that achieved some success. Chapter 7 discusses the extensions made to several of the GA-based attacks, as well as new work attempted.

Finally, Chapter 8 discusses the overall conclusions drawn from this work, as well as any other directions that need further investigation.

Chapter 2

Genetic Algorithms

The genetic algorithm is a search algorithm based on the mechanics of natural selection and natural genetics [12]. As summarized by Tomassini [29], the main idea is that in order for a population of individuals to adapt to some environment, it should behave like a natural system. This means that survival and reproduction of an individual is promoted by the elimination of useless or harmful traits and by rewarding useful behavior. The genetic algorithm belongs to the family of evolutionary algorithms, along with genetic programming, evolution strategies, and evolutionary programming. Evolutionary algorithms can be considered as a broad class of stochastic optimization techniques. An evolutionary algorithm maintains a population of candidate solutions for the problem at hand. The population is then evolved by the iterative application of a set of stochastic operators. The set of operators usually consists of mutation, recombination, and selection or something very similar. Globally satisfactory, if sub-optimal, solutions to the problem are found in much the same way as populations in nature adapt to their surrounding environment.

Using Tomassini's terms [29], genetic algorithms (GAs) consider an optimization problem as the environment where feasible solutions are the individuals living in that environment. The degree of adaptation of an individual to its environment is the counterpart of the fitness function evaluated on a solution. Similarly, a set of feasible solutions takes the place of a population of organisms. An individual is a string of binary digits or some other set of symbols drawn from a finite set. Each encoded individual in the population may be viewed as a representation of a particular solution to a problem.

In general, a genetic algorithm begins with a randomly generated set of individuals. Once the initial population has been created, the genetic algorithm enters a loop [29]. At the end of each iteration, a new population has been produced by applying a certain number of stochastic operators to the previous population. Each such iteration is known as a generation [29].

A selection operator is applied first. This creates an intermediate population of n "parent" individuals. To produce these "parents", n independent extractions of an individual from the old population are performed [29]. The probability of each individual being extracted should be (linearly) proportional to the fitness of that individual. This means that above average individuals should have more copies in the new population, while below average individuals should have few to no copies present, i.e., a below average individual risks extinction.

Once the intermediate population of "parents" (those individuals selected for reproduction) has been produced, the individuals for the next generation will be created through the application of a number of reproduction operators [29]. These operators can involve one or more parents. An operator that involves just one parent, simulating asexual reproduction, is called a mutation operator. When more than one parent is involved, sexual reproduction is simulated, and the operator is called recombination [29]. The genetic algorithm uses two reproduction operators - crossover and mutation.

To apply a crossover operator, parents are paired together. There are several different types of crossover operators, and the types available depend on what representation is used for the individuals. For binary string individuals, one-point, two-point, and uniform crossover are often used. For permutation or order-based individuals, order, partially mapped, and cycle crossover are options. The one-point crossover means that the parent individuals exchange a random prefix when creating the child individuals. Two-point crossover is an exchange of a random substring, and uniform crossover takes each bit in the child arbitrarily from either parent. Order and partially mapped crossover are similar to two-point crossover in that two cut points are selected. For order crossover, the section between the first and second cut points is copied from the first parent to the child [28]. The remaining places are filled using elements not occurring in this section, in the order that they occur in the second parent starting from the second cut point and wrapping around as needed. For partially mapped crossover, the section between the two cut points defines a series of swapping operations to be performed on the second parent [28]. Cycle crossover satisfies two conditions - every position of the child must retain a value found in the corresponding position of a parent, and the child must be a valid permutation. Each cycle, a random parent is selected. For example [28]:

- Parent 1 is (h k c e f d b l a i g j) and parent 2 is (a b c d e f g h i j k l)
- For position #1 in the child, ‘h’ is selected
- Therefore, ‘a’ cannot be selected from parent 2 and must be chosen from parent 1
- Since ‘a’ is above ‘i’ in parent 2, ‘i’ must also be selected from parent 1
- So, if ‘h’ is selected from parent 1 for position #1, then ‘a’, ‘i’, ‘j’, and ‘l’ must also be selected from parent 1
- The positions of these elements are said to form a cycle, since selection continues until the first element (‘h’ here) is reached again

After crossover, each individual has a small chance of mutation. The purpose of the mutation operator is to simulate the effect of transcription errors that can happen with a very low probability when a chromosome is mutated [29]. A standard mutation operator for binary strings is bit inversion. Each bit in an individual has a small chance of mutating into its complement i.e. a ‘0’ would mutate into a ‘1’.

In principle, the loop of selection-crossover-mutation is infinite [29]. However, it is usually stopped when a given termination condition is met. Some common termination conditions are [29]:

1. A pre-determined number of generations have passed

2. A satisfactory solution has been found
3. No improvement in solution quality has taken place for a certain number of generations

The different termination conditions are possible since a genetic algorithm is not guaranteed to converge to a solution. The evolutionary cycle can be summarized as follows [29]:

```
$generation = 0$
seed population
**while** not (termination condition) **do**
    $generation = generation + 1$
    calculate fitness
    selection
    crossover
    mutation
**end while**
```

Typical application areas for genetic algorithms include:

- Function Optimization
- Graph Applications such as
  - Coloring
  - Paths
  - Circuit Layout
- Scheduling
- Satisfiability
- Game Strategies
- Chess Problems

- Computing Models
- Neural Networks
- Lens Design

Many of these application areas are concerned with problems which are hard to solve but have easily verifiable solutions. Another trait common to these application areas is the equation style of fitness function. Cryptography and cryptanalysis could be considered to meet these criteria. However, cryptanalysis is not closely related to the typical GA application areas and, subsequently, fitness equations are difficult to generate. This makes the use of a genetic algorithm approach to cryptanalysis rather unusual.

Chapter 3

Cryptography

In general, the genetic algorithm approach has only been used to attack fairly simple ciphers. Most of the ciphers attacked are considered to be classical, i.e. those created before 1950 A.D. [21] which have become well known over time. Ciphers attacked include:

- Monoalphabetic Substitution cipher
- Polyalphabetic Substitution cipher
- Permutation cipher
- Transposition cipher
- Merkle-Hellman Knapsack cipher
- Chor-Rivest Knapsack cipher
- Vernam cipher

3.1 Monoalphabetic Substitution cipher

The simplest cipher of those attacked is the monoalphabetic substitution cipher. This cipher is described as follows:

- Let the plaintext and the ciphertext character sets be the same, say the alphabet $\mathcal{Z}$

- Let the keys, $\mathcal{K}$, consist of all possible permutations of the symbols in $\mathcal{Z}$
- For each permutation $\pi \in \mathcal{K}$,
  1. define the encryption function $e_\pi(x) = \pi(x)$
  2. and define the decryption function $d_\pi(y) = \pi^{-1}(y)$, where π^{-1} is the inverse permutation to π.

For example, define $\mathcal{Z}$ to be the 26 letter English alphabet. Then, a random permutation π could be (plaintext characters are in lower case and ciphertext characters in upper case) [27]:

a	b	c	d	e	f	g	h	i	j	k	l	m
X	N	Y	A	H	P	O	G	Z	Q	W	B	T

n	o	p	q	r	s	t	u	v	w	x	y	z
S	F	L	R	C	V	M	U	E	K	J	D	I

This defines the encryption function like so - $e_\pi(a) = X$, $e_\pi(b) = N$, etc. The decryption function, $d_\pi(y)$ is the inverse permutation, and can be obtained by switching the rows. π^{-1} is, therefore:

A	B	C	D	E	F	G	H	I	J	K	L	M
d	l	r	y	v	o	h	e	z	x	w	p	t

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
b	g	f	j	q	n	m	u	s	k	a	c	i

Using the encryption function, a plaintext of 'plain' becomes a ciphertext of 'LBXZS'. A ciphertext of 'YZLGHC' would decrypt to 'cipher'.

Monoalphabetic substitution is easy to recognize, since the frequency distribution of the character set is unchanged [21]. The frequency of individual characters changes, of course. This attribute allows associations between ciphertext and plaintext characters, i.e. if the most frequent plaintext character X occurred ten times, the ciphertext character X maps to will appear ten times.

3.2 Polyalphabetic Substitution cipher

This cipher is a more complex version of the substitution cipher. Instead of mapping the plaintext alphabet onto ciphertext characters, different substitution mappings are used on different portions of the plaintext [21]. The result is called polyalphabetic substitution. The simplest case consists of using different alphabets sequentially and repeatedly, so that the position of each plaintext character determines which mapping is applied to it. Under different alphabets, the same plaintext character is encrypted to different ciphertext characters, making frequency analysis harder.

This cipher has the following properties (assuming m mappings):

- The key space $\mathcal{K}$ consists of all ordered sets of m permutations
- These permutations are represented as $(k_1, k_2, \ldots, k_m)$
- Each permutation k_i is defined on the alphabet in use
- Encryption of the message $x = (x_1, x_2, \ldots, x_m)$ is given by
- $e_k(x) = k_1(x_1)k_2(x_2)\ldots k_m(x_m)$, assuming the key $k = (k_1, k_2, \ldots, k_m)$
- The decryption key associated with $e_k(x)$ is $d_k(y) = (k_1^{-1}, k_2^{-1}, \ldots, k_m^{-1})$.

The encryption and decryption functions could also be written as follows (assuming that the addition/subtraction operations occur modulo the size of the alphabet):

- $e_K(x_1, x_2, \ldots, x_m) = (x_1 + k_1, x_2 + k_2, \ldots, x_m + k_m)$
- $d_K(y_1, y_2, \ldots, y_m) = (y_1 - k_1, y_2 - k_2, \ldots, y_m - k_m)$

The most common polyalphabetic substitution cipher is the Vigenère cipher. This cipher encrypts m characters at a time, making each plaintext character equivalent to m alphabetic characters. A example of the Vigenère cipher is:

- Suppose $m = 6$
- Using the mapping $A \leftrightarrow 0, B \leftrightarrow 1, \ldots, Z \leftrightarrow 25$
- If the keyword $\mathcal{K}$ = CIPHER, numerically $(2, 8, 15, 7, 4, 17)$
- Suppose the plaintext is the string 'cryptosystem'
- Convert the plaintext elements to residues modulo 26 - 2 17 24 15 19 14 18 24 18 19 4 12
- Write the plaintext elements in groups of 6 and add the keyword modulo 26
-

2	17	24	15	19	14	18	24	18	19	4	12
2	8	15	7	4	17	2	8	15	7	4	17
4	25	13	22	23	5	20	6	7	0	8	3

- The alphabetic equivalent of the ciphertext string is: EZNWXFUGHAID.
- To decrypt, the same keyword is used but is subtracted modulo 26 from the ciphertext

3.3 Permutation/Transposition ciphers

Although simple transposition ciphers change the dependencies between consecutive characters, they are fairly easily recognized since the frequency distribution of the characters is preserved [21]. This is true for both ciphers considered, the permutation cipher and the columnar transposition cipher. Both ciphers apply the same principles, but differ in how the transformation is applied. The permutation cipher is applied to blocks of characters while the columnar transposition cipher is applied to the entire text at once.

3.3.1 Permutation cipher

The idea behind a permutation cipher is to keep the plaintext characters unchanged, but alter their positions by rearrangement using a permutation [27].

This cipher is defined as:

- Let m be a positive integer, and $\mathcal{K}$ consist of all permutations of $\{1, \ldots, m\}$
- For a key (permutation) $\pi \in \mathcal{K}$, define:
- The encryption function $e_\pi(x_1, \ldots, x_m) = (x_{\pi(1)}, \ldots, x_{\pi(m)})$
- The decryption function $d_\pi(y_1, \ldots, y_m) = (y_{\pi^{-1}(1)}, \ldots, y_{\pi^{-1}(m)})$

A small example, assuming $m = 6$, and the key is the permutation π:

i	1	2	3	4	5	6
$\pi(i)$	3	5	1	6	4	2

The first row is the value of i, and the second row is the corresponding value of $\pi(i)$. The inverse permutation, π^{-1} is constructed by interchanging the two rows, and rearranging the columns so that the first row is in increasing order, Therefore, π^{-1} is:

x	1	2	3	4	5	6
$\pi^{-1}(x)$	3	6	1	5	2	4

If the plaintext given is ‘apermutation’, it is first partitioned into groups of m letters (‘apermu’ and ‘tation’) and then rearranged according to the permutation π. This yields ‘EMAURP’ and ‘TOTNIA’, making the ciphertext equal ‘EMAURPTOTNIA’. The ciphertext is decrypted in a similar process, using the inverse permutation π^{-1}.

3.3.2 Transposition cipher

Another type of cipher in this category is the columnar transposition cipher. As described in [24], this cipher consists of writing the plaintext into a rectangle with a prearranged number of columns and transcribing it vertically by columns to yield the ciphertext. The key is a prearranged sequence of numbers which determines both the width of the inscription rectangle and the order in which to transcribe the columns [24]. This key is usually derived from an agreed keyword by assigning numbers to the individual letters according to their alphabetical order.

For example, if the keyword is SORCERY, the resulting numerical key would be 6 3 4 1 2 5 7 [24], where numbers are assigned to the keyword letters according to the alphabetical order of the letters. If we wish to encipher the message LASER BEAMS CAN BE MODULATED TO CARRY MORE INTELLIGENCE THAN RADIO WAVES, it would first be written out on a width of 7 under the numerical key. As is traditional, white spaces and punctuation are ignored in this and all ciphertexts used. The rectangle would look like this:

S	O	R	C	E	R	Y
6	3	4	1	2	5	7
L	A	S	E	R	B	E
A	M	S	C	A	N	B
E	M	O	D	U	L	A
T	E	D	T	O	C	A
R	R	Y	M	O	R	E
I	N	T	E	L	L	I
G	E	N	C	E	T	H
A	N	R	A	D	I	O
W	A	V	E	S	Q	R

Since the message length was not a multiple of the keyword length, the message does not fill the entire rectangle. Deciphering is simplified if the last row contains no blank cells, so two dummy letters are added (here, Q and R) to make the rectangle completely filled [24]. Enciphering consists of reading the ciphertext out vertically in the order of the numbered columns. It can be written in groups of five letters at the same time, if so desired [24]. The ciphertext is, therefore, ECDTM ECAER AUOOL EDSAM MERNE NASSO DYTNR VBNLC RLTIQ LAETR IGAWE BAAEI HOR.

The decipherer would count the number of letters in the message (63) and determine the length of the inscription rectangle by dividing this value by the length of the keyword. This makes the inscription rectangle 7×9 in this case. The numerical key is then written above the rectangle and the ciphertext characters entered into the diagram in the order of the key numbers [24]. The plaintext is read off by rows, as it was originally entered.

3.4 Knapsack ciphers

Knapsack public-key encryption systems are based on the subset sum problem, an NP-complete problem. Given in this problem are a finite set $S \subset \mathbb{N}^+$ ($\mathbb{N}^+$ is the set of natural numbers $\{1, 2, \dots\}$), and a target $t \in \mathbb{N}^+$. The question is whether there exists a subset $S' \subseteq S$ whose elements sum to t [9]. For example:

- If $S = \{1, 2, 7, 14, 49, 98, 343, 686, 2409, 2793, 16808, 17206, 117705, 117993\}$
- And $t = 138457$
- Then the subset $S' = \{1, 2, 7, 98, 343, 686, 2409, 17206, 117705\}$ is a solution.

The basic idea of a knapsack cipher is to select an instance of the subset sum problem that is easy to solve, and then disguise it as an instance of the general subset sum problem which is hopefully difficult to solve [21]. The original (easy) knapsack set can serve as the private key, while the transformed knapsack set serves as the public key.

The Merkle-Hellman knapsack cipher is important as it was the first implementation of a public-key encryption scheme. Many variations on it have been suggested but most, including the original, have been demonstrated to be insecure [21]. A notable exception to this is the Chor-Rivest knapsack system [21].

3.4.1 Merkle-Hellman Knapsack cipher

The Merkle-Hellman knapsack cipher attempts to disguise an easily solved instance of the subset sum problem, called a superincreasing subset sub problem, by modular multiplication and a permutation [21]. A superincreasing sequence is a sequence $(b_1, b_2, \ldots, b_n)$ of positive integers with the property that $b_i > \sum_{j=1}^{i-1} b_j$, for each $i, 2 \leq i \leq n$.

The integer n is a common system parameter. $b_1, b_2, \ldots, b_n$ is the superincreasing sequence and M is the modulus, selected such that $M > b_1 + b_2 + \ldots + b_n$. W is a random integer, such that $1 \leq W \leq M - 1$ and $\gcd(W, M) = 1$. π is a random permutation of the integers $\{1, 2, \ldots, n\}$. The public key of A is $(a_1, a_2, \ldots, a_n)$, where $a_i = W b_{\pi(i)} \mod M$, and the private key of A is $(\pi, M, W, (b_1, b_2, \ldots, b_n))$.

The steps in the transmission of a message m are as follows (B is the sender, A is the receiver) [21]:

1. Encryption - B should do the following:
   - Obtain A's authentic public key $(a_1, a_2, \ldots, a_n)$
   - Represent the message m as a binary string of length n,
   - $m = m_1 m_2 \ldots m_n$
   - Compute the integer $c = m_1 a_1 + m_2 a_2 + \ldots + m_n a_n$
   - Send the ciphertext c to A
2. Decryption - To recover the plaintext m from c, A should do the following:
   - Compute $d = W^{-1} c \mod M$
   - By solving a superincreasing subset sum problem, find the integers $r_1, r_2, \ldots, r_n$, $r_i \in \{0, 1\}$ such that $d = r_1 b_1 + r_2 b_2 + \ldots + r_n b_n$.
   - The message bits are $m_i = r_{\pi(i)}$, $i = 1, 2, \ldots, n$.

A small example is as follows [21]:

1. Let $n = 6$
2. Entity A chooses the superincreasing sequence $(12, 17, 33, 74, 157, 316)$
3. $M = 737$
4. $W = 635$
5. The permutation π of $\{1, 2, 3, 4, 5, 6\}$ is defined as:
6. $\pi(1) = 3, \pi(2) = 6, \pi(3) = 1, \pi(4) = 2, \pi(5) = 5, \pi(6) = 4$
7. A's public key is the knapsack set $(319, 196, 250, 477, 200, 559)$
8. A's private key is $(\pi, M, W, (12, 17, 33, 74, 157, 316))$
9. To encrypt the message $m = 101101$, B computes c as:
10. $c = 319 + 250 + 477 + 559 = 1605$
11. and sends it to A

1. To decrypt, A computes
2. $d = W^{-1}c \mod M = 136$
3. and solves the superincreasing subset sum problem of:
4. $136 = 12r_1 + 17r_2 + 33r_3 + 74r_4 + 157r_5 + 316r_6$
5. to get $136 = 12 + 17 + 33 + 74$
6. Therefore, $r_1 = 1, r_2 = 1, r_3 = 1, r_4 = 1, r_5 = 0, r_6 = 0$
7. Applying the permutation π produces the message bits:
8. $m_1 = r_3 = 1, m_2 = r_6 = 0, m_3 = r_1 = 1, m_4 = r_2 = 1, m_5 = r_5 = 0, m_6 = r_4 = 1$

3.4.2 Chor-Rivest Knapsack cipher

The Chor-Rivest cipher is the only known knapsack public-key system that does not use some form of modular multiplication to disguise an easy subset sum problem [21]. This cipher is by far the most sophisticated of those attacked by a genetic algorithm. When the parameters of this system are carefully chosen, and no information is leaked, there is no feasible attack on the Chor-Rivest system. It can not be covered in detail in a few pages, so only a summary of the system will be given. Further number theoretical algorithms are needed for the set-up and operation of the cipher.

Most of the parameters are based on a finite field $\mathbb{F}_q$, of characteristic p, where $q = p^h, p \geq h$ and for which the discrete logarithm problem is feasible [21]. $f(x)$ is a random monic irreducible polynomial of degree h over $\mathbb{Z}_p$. The elements of $\mathbb{F}_q$ are represented as polynomials in $\mathbb{Z}_p[x]$ of degree less than h, with multiplication done modulo $f(x)$. $g(x)$ is a random primitive element of $\mathbb{F}_q$. For each ground field element $i \in \mathbb{Z}_p$, the discrete logarithm $a_i = \log_{g(x)}(x + i)$ of the field element $(x + i)$ to the base $g(x)$ must be found. The random permutation π is on the set of integers $\{0, 1, 2, \ldots, p - 1\}$, and the random integer d is selected such that $0 \leq d \leq p^h - 2$. c_i is then computed as $c_i = (a_{\pi(i)} + d) \bmod (p^h - 1), 0 \leq i \leq p - 1$. A's public key is $((c_0, c_1, \ldots, c_{p-1}), p, h)$ and its private key is $(f(x), g(x), \pi, d)$.

A message m is transmitted as follows (B is the sender, A is the receiver) [21]:

1. Encryption - B should do the following:

   - Obtain A's authentic public key $((c_0, c_1, \ldots, c_{p-1}), p, h)$
   - Represent the message m as a binary string of length $\lfloor \lg \binom{p}{h} \rfloor$, where $\binom{p}{h}$ is a binomial coefficient
   - Consider m as the binary representation of an integer
   - Transform this integer into a binary vector $M = (M_0, M_1, \ldots, M_{p-1})$ of length p having exactly h 1's as follows:

     (a) Set $l \leftarrow h$

    (b) For i from 1 to p do the following:

    (c) If $m \geq \binom{p-i}{l}$ then set $M_{i-1} \leftarrow 1$, $m \leftarrow m - \binom{p-i}{l}$, $l \leftarrow l - 1$

    (d) Otherwise, set $M_{i-1} \leftarrow 0$

    (e) Note: $\binom{n}{0} = 1$ for $n \geq 0$, $\binom{0}{l} = 0)$ for $l \geq 1$

- Compute $c = \sum_{i=0}^{p-1} M_i c_i \mod (p^h - 1)$
- Send the ciphertext c to A

2. Decryption - To recover the plaintext m from c, A should do the following:

- Compute $r = (c - hd) \mod (p^h - 1)$
- Compute $u(x) = g(x)^r \mod f(x)$
- Compute $s(x) = u(x) + f(x)$, a monic polynomial of degree h over $\mathbb{Z}_p$ ($\mathbb{Z}_p$ is the set of integers $\{0, \ldots, p-1\}$. Addition, subtraction, and multiplication in $\mathbb{Z}_p$ are performed modulo p.)
- Factor $s(x)$ into linear factors over $\mathbb{Z}_p : s(x) = \prod_{j=1}^{h} (x + t_j)$, where $t_j \in \mathbb{Z}_p$
- Compute a binary vector $M = (M_0, M_1, \ldots, M_{p-1})$ as follows:

    (a) The components of M that are 1 have indices $\pi^{-1}(t_j)$, $1 \leq j \leq h$.

    (b) The remaining components are 0.

- The message m is recovered from M as follows:

    (a) Set $m \leftarrow 0, l \leftarrow h$

    (b) For i from 1 to p do the following:

    (c) if $M_{i-1} = 1$, then set $m \leftarrow m + \binom{p-i}{l}$ and $l \leftarrow l - 1$

An example using artificially small parameters would go as follows [21]:

1. Entity A selects $p = 7$ and $h = 4$

2. Entity A selects the irreducible polynomial $f(x) = x^4 + 3x^3 + 5x^2 + 6x + 2$ of degree 4 over $\mathbb{Z}_7$. The elements of the finite field $\mathbb{F}_{7^4}$ are represented as polynomials in $\mathbb{Z}_7[x]$ of degree less than 4, with multiplication performed modulo $f(x)$.

3. Entity A selects the random primitive element $g(x) = 3x^3 + 3x^2 + 6$

4. Entity A computes the following discrete logarithms:

   - $a_0 = \log_{g(x)}(x) = 1028$
   - $a_1 = \log_{g(x)}(x+1) = 1935$
   - $a_2 = \log_{g(x)}(x+2) = 2054$
   - $a_3 = \log_{g(x)}(x+3) = 1008$
   - $a_4 = \log_{g(x)}(x+4) = 379$
   - $a_5 = \log_{g(x)}(x+5) = 1780$
   - $a_6 = \log_{g(x)}(x+6) = 223$

5. Entity A selects the random permutation π on $\{0, 1, 2, 3, 4, 5, 6\}$ defined by $\pi(0) = 6, \pi(1) = 4, \pi(2) = 0, \pi(3) = 2, \pi(4) = 1, \pi(5) = 5, \pi(6) = 3$

6. Entity A selects the random integer $d = 1702$

7. Entity A computes:

   - $c_0 = (a_6 + d) \mod 2400 = 1925$
   - $c_1 = (a_4 + d) \mod 2400 = 2081$
   - $c_2 = (a_0 + d) \mod 2400 = 330$
   - $c_3 = (a_2 + d) \mod 2400 = 1356$
   - $c_4 = (a_1 + d) \mod 2400 = 1237$
   - $c_5 = (a_5 + d) \mod 2400 = 1082$
   - $c_6 = (a_3 + d) \mod 2400 = 310$

8. Entity A's public key is $((c_0, c_1, c_2, c_3, c_4, c_5, c_6), p = 7, h = 4)$, and its private key is $(f(x), g(x), \pi, d)$.

9. To encrypt a message $m = 22$ for A, entity B does the following:

   (a) Obtains A's authentic public key $((c_0, c_1, c_2, c_3, c_4, c_5, c_6), p, h)$

   (b) Represents m as a binary string of length 5: $m = 10110$ (Note that $\lfloor \lg \binom{7}{4} \rfloor = 5$).

   (c) Transforms m to the binary vector $M = (1, 0, 1, 1, 0, 0, 1)$ of length 7

   (d) Computes $c = (c_0 + c_2 + c_3 + c_6) \mod 2400 = 1521$

   (e) Sends $c = 1521$ to A

10. To decrypt the ciphertext $c = 1521$, A does the following:

   (a) Computes $r = (c - hd) \mod 2400 = 1913$

   (b) Computes $u(x) = g(x)^(1913) \mod f(x) = x^3 + 3x^2 + 2x + 5$

   (c) Computes $s(x) = u(x) + f(x) = x^4 + 4x^3 + x^2 + x$

   (d) Factors $s(x) = x(x + 2)(x + 3)(x + 6)$ (so $t_1 = 0, t_2 = 2, t_3 = 3, t_4 = 6$).

   (e) The components of M that are 1 have indices $\pi^{-1}(0) = 2, \pi^{-1}(2) = 3, \pi^{-1}(3) = 6, \pi^{-1}(6) = 0$. Hence, $M = (1, 0, 1, 1, 0, 0, 1)$

   (f) Transforms M to the integer $m = 22$, recovering the plaintext

To illustrate how small the above parameters are, the parameters originally suggested for the Chor-Rivest system are [21]:

- $p = 197$ and $h = 24$
- $p = 211$ and $h = 24$
- $p = 3^5$ and $h = 24$
- $p = 2^8$ and $h = 25$

3.5 Vernam cipher

The Vernam cipher is a stream cipher defined on the alphabet $\mathcal{A} = \{0, 1\}$. A binary message $m_1 m_2 \ldots m_t$ is operated on by a binary key string $k_1 k_2 \ldots k_t$ of the same length to produce a ciphertext string $c_1 c_2 \ldots c_t$ where $c_i = m_i \oplus k_i, 1 \leq i \leq t$ [21]. If the key string is randomly chosen and never used again, this cipher is called a one-time system or one-time pad.

The one-time pad can be shown to be theoretically unbreakable [21]. If a cryptanalyst has a ciphertext string $c_1 c_2 \ldots c_t$ encrypted using a random, non-reused key string, the cryptanalyst can do no better than guess at the plaintext being any binary string of length t. It has been proven that an unbreakable Vernam system requires a random key of the same length as the message [21].

3.6 Comments

The above ciphers comprise all those cryptosystems attacked using a genetic algorithm in a fairly exhaustive search of the literature. The newest of these systems are the knapsack ciphers, with the Chor-Rivest system published in 1985, and the Merkle-Hellman system published in 1978 [21]. The other ciphers attacked are all systems older than that. In some cases, centuries older. None of the modern ciphers, such as DES, AES, RSA, or Elliptic Curve, were even attempted using a genetic algorithm based approach. The only ciphers that are considered even remotely mathematically difficult are the knapsack systems, and even they are not true number theoretical protocols. This lack makes many of the genetic algorithm attacks have little importance in the field of cryptanalysis.

Chapter 4

Relevant Prior Works

While there have been many papers written on genetic algorithms and their application to various problems, there are relatively few papers that apply genetic algorithms to cryptanalysis. The earliest papers that use a genetic algorithm approach for a cryptanalysis problem were written in 1993, almost twenty years after the primary paper on genetic algorithms by John Holland [12] [14]. These papers focus on the simplest classical and modern ciphers, and are cited frequently by later works.

Figures 4.1 through 4.4 show which ciphers are attacked by which authors.

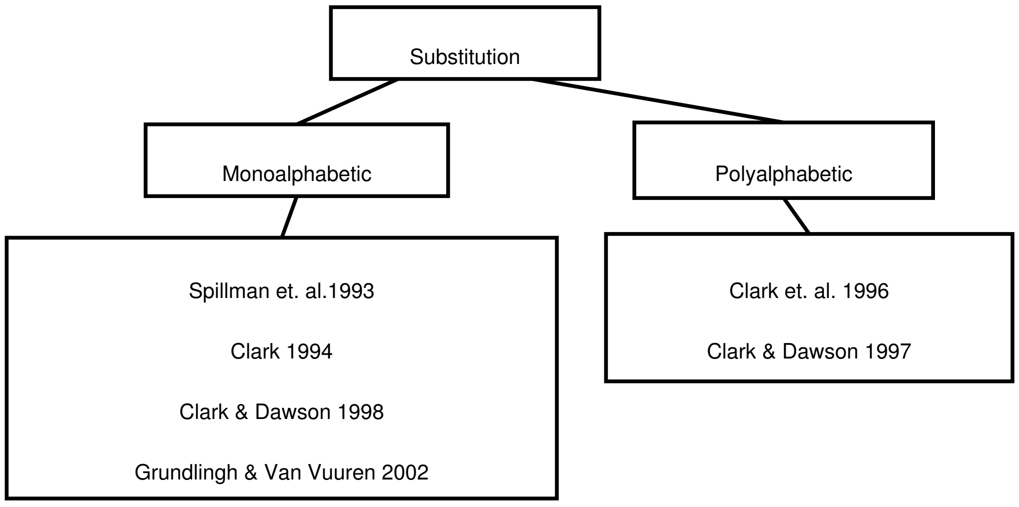

Figure 4.1: Substitution Cipher Family and Attacking Papers

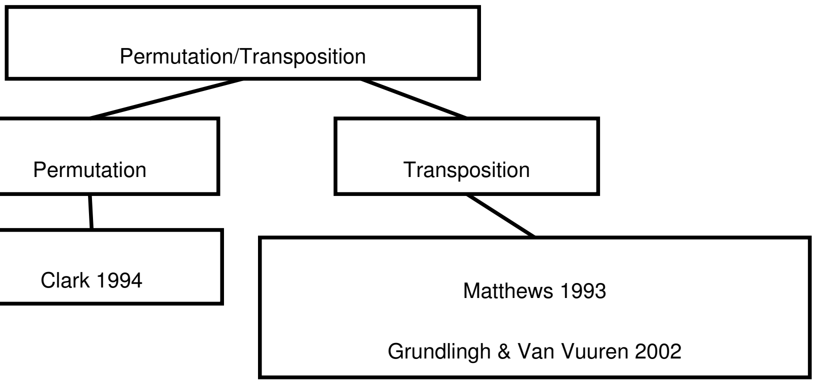

Figure 4.2: Permutation Cipher Family and Attacking Papers

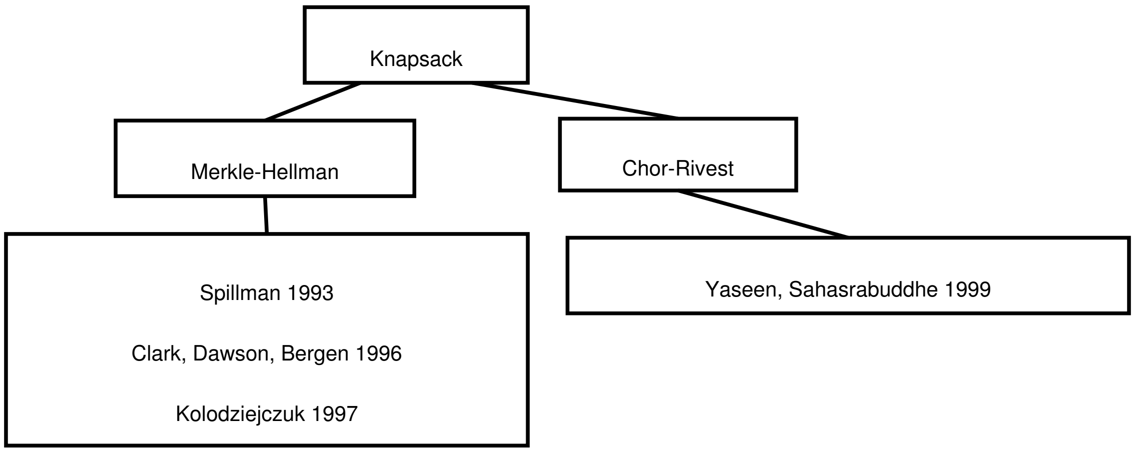

Figure 4.3: Knapsack Cipher Family and Attacking Papers

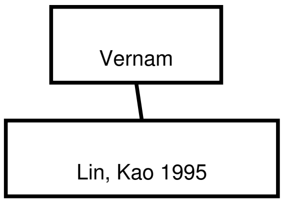

Figure 4.4: Vernam Cipher Family and Attacking Paper

4.1 Substitution: Spillman et al. - 1993

The first paper published in 1993 by Spillman, Janssen, Nelson and Kepner [26], focuses on the cryptanalysis of a simple substitution cipher using a genetic algorithm. This paper selects a genetic algorithm approach so that a directed random search of the key space can occur. The paper begins with a review of genetic algorithms, covering the genetic algorithm life cycle, and the systems needed to apply a genetic algorithm. The systems are considered to be a key representation, a mating scheme, and a mutation scheme. The selected systems are then discussed. The key representation is an ordered list of alphabet characters, where the first character in the list is the plaintext character for the most frequent ciphertext character, the second character is the plaintext character for the second most frequent ciphertext character, and so on. The mating scheme randomly selects two keys for mating, weighted in favor of keys with a high fitness value. The two parent keys create two child keys by selecting from each slot in the parents the more frequent ciphertext character. The first child key is created by scanning left to right, and the second by scanning right to left. If the more frequent character is already in the child, then the character from the other parent is selected. The mutation scheme consists of swapping two characters in the key string. The fitness function uses unigrams and digrams, and is as follows:

$$Fitness = (1 - \sum_{i=1}^{26}\{|SF[i] - DF[i]| + \sum_{j=1}^{26}|SDF[i,j] - DDF[i,j]|\}/4)^8 \qquad (4.1.1)$$

The function represents a normalized error value. Larger fitness values represent smaller errors. Sensitivity to small differences is increased by raising the result to the 8th power, while sensitivity to large differences is decreased by the division of the summation terms by 4. $SF[i]$ is the standard frequency of character i in English plaintext, while $DF[i]$ is the measured frequency of the decoded character i in the ciphertext. This makes the first summation term equal to the sum of the differences between each character's standard frequency and the frequency of the ciphertext character that decodes to that character. SDF is the standard frequency for a digram and DDF the measured frequency. When the measured and standard frequencies are the same, the summation terms equal zero, making the fitness value equal one. This fitness function is bounded below by zero, but cannot evaluate to zero. Since high fitness values are more important, this does not effect the overall result. The complete algorithm used in this work is:

1. Generate a random population of key strings
2. Determine a fitness value for each key in the population
3. Do a biased random selection of parents
4. Apply the crossover operation to the parents
5. Apply the mutation process to the children
6. Scan the new population of key strings and update the list of the 10 “best” keys seen

The process stops after a fixed number of generations, at which point, the best keys are used to decipher the ciphertext. Results are given for 100 generation runs of the algorithm, using populations of 10 and 50 keys, and mutation rates of 0.05, 0.10, 0.20, and 0.40. In the smaller population size, the exact key is found after searching less than $4x10-23$ of the key space, or after the examination of about 1000 keys. In the larger population size, only about 20 generations are needed to examine 1000 keys. The best mutation rates for this experiment were those less than 0.20.

4.1.1 Comments

Overall, this paper is a good choice for re-implementation and comparison. It is an early work, so does not compare its results to others, but provides a decent level of result reporting and discussion, making comparison possible. It presents the basic genetic algorithm concepts clearly, and uses examples and diagrams to clarify important points.

- Typical results:
  - The exact key is not always found

    - Visual inspection of the plaintext is required most of the time to determine the exact key

- Re-implementation approach:
    - The same parameters are re-used
    - Additional parameters are used to expand the range of parameter values
- Observed cryptanalytic usefulness:
    - None - no exact keys found in testing
- Comment on result discrepancy:
    - Different measures of success were used

4.2 Transposition: Matthews - 1993

The second paper published in 1993 [20], by R. A. J. Matthews, uses an order-based genetic algorithm to attack a simple transposition cipher. The paper begins by discussing the archetypal genetic algorithm. This includes the typical genetic algorithm components, as well as the two characteristics a problem must have in order for it to be attackable by a genetic algorithm. The first characteristic is that a partially-correct chromosome must have a higher fitness than an incorrect chromosome. The second characteristic is an encoding for the problem that allows genetic operators to work. The next section of the paper discusses the cryptanalytic use of the genetic algorithm. It concludes that substitution, transposition, permutation, and other ciphers are attackable by a genetic algorithm, while ciphers such as DES (Data Encryption Standard), which is a Feistel cipher, are not attackable.

The cipher selected for attack in this work is a classical two-stage transposition cipher. Here, a key of length N takes the form of a permutation of the integers 1 to N. The plaintext, L characters long, is written beneath the key to form a matrix N characters wide by (at least) $L \mod N$ characters deep. The second stage consists of reading the text off the columns in the order indicated by the integers in the key. Decryption consists of writing down the key, determining the length of the columns, writing the ciphertext back into the matrix in sequence, and reading across the columns. Breaking this cipher is typically a two-stage process as well. The stages are determining the length of the transposition sequence (N), and then finding the permutation of the N integers.

The genetic algorithm system used to attack the transposition cipher is known as GENALYST. It incorporates all the standard genetic algorithm features, with the addition of the items needed for an order-based approach. The GENALYST system uses a fitness scoring system based on the number of times common English digrams and trigrams appear in the decrypted text. Ten digram and trigrams were used - TH, HE, IN, ER, AN, ED, THE, ING, AND, and EEE. The presence of EEE resulted in a subtraction of fitness points, as it practically never appears in English. This process was checked for correlation between fitness value and correctness of chromosome, to improve confidence in GENALYST. GENALYST selects the fittest chromosomes using a roulette wheel approach based on normalized fitness values. This means that the probability of a chromosome getting selected is proportional to its normalized fitness value. “Elitism” is present to ensure that the fittest or most elite chromosomes found so far are always included in the breeding pool. After selection occurs, breeding is done with a proportion of the remaining chromosomes. The proportion used decreases linearly as the search continues, in order to help optimize the effectiveness of GENALYST. Breeding is done using the position based crossover method for order-based genetic algorithms. In this method, a random bit string of the same length as the chromosomes is used. The random bit string indicates which elements to take from parent #1 by a 1 in the string, and the elements omitted from parent #1 are added in the order they appear in parent #2. The second child is produced by taking elements from parent #2 indicated by a 0 in the string, and filling in from parent #1 as before. Two possible mutation types may then be applied. The first type is a simple two-point mutation where two random chromosome elements are swapped. The second type is a shuffle mutation, where the permutation is shifted forward by a random number of places.

A test procedure is given to ensure that test texts have a true fitness value that is representative of English text. This is calculated mathematically, based on digram/trigram percentage frequency in the text, the fitness score given by GENALYST to that di- or tri-gram, and the summation over all di- and tri-grams checked. The text length is also important, as it is easier to attack texts with a length that is an integer multiple of the key length. Based on these considerations, a test text with a length of 181 characters was selected. This length is prime, but close to multiples of 6, 9, 10, and 12. This is a worst-case scenario since not all column lengths are the same, and multiple possibilities for the key length are present. A Monte Carlo search is run on the same text so that the genetic algorithm results can be compared to the results from a traditional search technique. In a Monte Carlo search, a series of guesses is made to determine the correct key, and the most elite chromosome is kept.

The breaking of a transposition cipher has two parts: finding the correct keylength, and finding the correct permutation of the key. GENALYST was tested on both parts. For the first part, the test text was encrypted using a target key. GENALYST was then used to find the best decryption of the text, using seven different keylengths in the range of 6 to 12, which includes the length of the target key. The goal of this section was to see if the fittest of the seven chromosomes had the same length as the target key. The population size used was 20, and the number of generations was 25. The probability of crossover started at 0.8 and decreased to 0.5. Mutation probability started at 0.1 for both point mutation and shuffle mutation, increasing to 0.5 for point mutation and to 0.8 for shuffle mutation. The experiment was run five times each for three target keylengths, $K = 7, 9, 11$. GENALYST was successful in 13 of the 15 runs (87%). However, the random Monte Carlo algorithm performed equally well in this task, also producing a success rate of 87%. For the second part, the same parameters of population size and number of generations were used. GENALYST managed to completely break the cipher in one of the runs for $K = 7$ and one of the runs for $K = 9$. Overall, GENALYST involves little more computational effort than the Monte Carlo algorithm, since the single largest task, fitness rating, is present in both.

Therefore, a genetic algorithm approach is useful even when only a small advantage is produced. Using a larger gene pool and number of generations boosts GENALYST's performance significantly over the performance of Monte Carlo. The greatest improvement in GENALYST's performance comes from hybridizing the algorithm with other techniques, such as perming.

4.2.1 Comments

Overall, this is one of the best reference papers available. It is an early work, but is very complete. It covers background material and problem considerations well, and has a very balanced approach to considering the genetic algorithm as a possible solution. The parameters are clearly reported, as are the algorithms used. This is an excellent candidate for re-implementation and comparison.

- Typical results:
  - Keylength correctly determined 87% of the time
  - Correct key found 13.33% of the time
- Re-implementation approach:
  - Different text lengths used, same parameters re-used
  - Additional parameters are used to expand the range of parameter values
- Observed cryptanalytic usefulness:
  - Low - low decryption percentage
- Comment on result discrepancy:
  - Percentages reported are based on number of tests and different numbers of tests were used

4.3 Merkle-Hellman Knapsack: Spillman - 1993

The third paper published in 1993, by R. Spillman [25], applies a genetic algorithm approach to a Merkle-Hellman knapsack system. This paper considers the genetic algorithm as simply another possibility for the cryptanalysis of knapsack ciphers. A knapsack cipher is based on the NP-complete problem of knapsack packing, which is an application of the subset sum problem. The problem statement is: "Given n objects each with a known volume and a knapsack of fixed volume, is there a subset of the n objects which exactly fills the knapsack ?". The idea behind the knapsack cipher is that finding the sum is hard, and this fact can be used to protect an encoding of information. There must also be a decoding algorithm that is relatively easy for the intended recipient to apply. An easy decoding algorithm can be made by using the easy knapsacks. An easy knapsack consists of a sequence of numbers that is superincreasing, i.e., each number is greater than the sum of the previous numbers. However, an easy knapsack does not protect the data if the elements of the knapsack are known. A Merkle-Hellman knapsack system converts an easy knapsack into a trapdoor knapsack. A trapdoor knapsack is not superincreasing, making it harder to attack. The Merkle-Hellman procedure to create a trapdoor knapsack sequence A is as follows:

1. Given a simple knapsack sequence $A' = (a'_1, \ldots, a'_n)$
2. Select an integer $u > 2a'_n$
3. Select another integer w such that $\gcd(u, w) = 1$
4. Find the inverse of w mod u
5. Construct the trapdoor sequence $A = wA'$ mod u

Once the easy knapsack has been converted into a trapdoor knapsack, both the easy knapsack and the value of w^{-1} must be known to decode the data. The target sum for the superincreasing sequence is calculated by multiplying the sum of the trapdoor sequence and

w^{-1} together and then reducing it modulo u. This system can become a public-key cipher, with the trapdoor sequence the public key. The private key would be the superincreasing sequence and the values of u, w, and w^{-1}.

The next section of the paper introduces genetic algorithms using binary chromosomes. The applicable processes of selection, mating, and mutation are described. Selection is a random process which is biased so that the chromosomes with higher fitness values are more likely to be selected. The mating process is a simple crossover operation, i.e., bits $s + 1$ to r of parent #1 are swapped with bits $s + 1$ to r of parent #2. Mutation consists of switching a bit in the chromosome to its complement, i.e., a 0 becomes a 1.

The three systems that need to be defined for a genetic algorithm application are: representation scheme, mating process, and mutation process. The representation used here is that of binary chromosomes, which represent the summation terms of the knapsack, i.e., a 1 in the chromosome indicates that that term should be included when calculating the knapsack sum. The evaluation function is determined once a representation scheme has been selected. Here, the function measures how close a given sum of terms is to the target sum of the knapsack. Other properties included for this experiment are:

- Function range between 0 and 1, with 1 indicating an exact match
- Chromosomes that produce a sum greater than the target should have a lower fitness, in general, than those that produce a sum less than the target
- It should be difficult to produce a high fitness value

The actual evaluation function used depends on the relationship between the target sum and the chromosome sum. The function is determined as follows:

1. Calculate the maximum difference that could occur between a chromosome and the target sum

$$MaxDiff = \max(Target, FullSum - Target) \tag{4.3.1}$$

where $FullSum$ is the sum of all components in the knapsack

2. Determine the Sum of the current chromosome

3. If $Sum \leq Target$ then

$$Fitness = 1 - \sqrt{|Sum - Target|/Target} \tag{4.3.2}$$

4. If $Sum > Target$ then

$$Fitness = 1 - \sqrt[6]{|Sum - Target|/MaxDiff} \tag{4.3.3}$$

As described above, the mating process is simple crossover. The mutation process consists of three different possibilities. About half the time, bits are randomly mutated as described above (switched to their complement). Next most likely is the swapping of a bit with a neighboring bit. The final possibility is a reversal of the bit order between two points.

A description of the complete algorithm is given, as well as results for one 15-item knapsack applied to a 5 character text. The characters are encoded as 8 bit ASCII characters. Each of the 5 sums (one per character) was attacked using the genetic algorithm, with an initial population of 20 random 15-bit binary strings.

4.3.1 Comments

This paper is reasonable for its length (about 10 pages) and year. A minimal level of experimentation is included but the results and parameters are decently reported. The introductory material and discussion are reasonable, although short. Re-implementation should not be difficult, and there are sufficient results to make a comparison possible. Overall, this paper is a reasonable choice for re-implementation and comparison.

- Typical results:
  - Used 5 ASCII characters, each of which was decrypted
- Re-implementation approach:

  - Not re-implemented

- Observed cryptanalytic usefulness:
  - None - the knapsack size is trivial

- Comment on result discrepancy:
  - No comment, not re-implemented

4.4 Substitution/Permutation: Clark - 1994

The only paper published in 1994, by Andrew Clark [4], includes the genetic algorithm as one of three optimization algorithms applied to cryptanalysis. The other two algorithms are tabu search and simulated annealing. The first section of the paper describes the two ciphers used - simple substitution and permutation. Next, suitability assessment is discussed. This section is where the fitness functions are developed. For the substitution cipher, the fitness function is

$$F_k = (\alpha \sum_{i \in \Lambda} |SF[i] - DF[i]| + \beta \sum_{i \in \Lambda} \sum_{j \in \Lambda} |SDF[i][j] - DDF[i][j]|) \tag{4.4.1}$$

Λ denotes the set of characters in the alphabet - $(A, B, C, \ldots, Z, space)$. $SF[i]$ is the relative frequency of character i in English, while $DF[i]$ is the relative frequency of the decoded character i in the message decrypted using the key k k. $SDF[i][j]$ is the relative frequency of the digram ij in English, while $DDF[i][j]$ is the relative frequency of that digram in the message decrypted using k. Varying α and β allows one to favor either the unigram or digram frequencies. For the permutation cipher, the same approach was used as in [20] - the scoring table for digrams and trigrams.

The next three sections introduce the basic concepts of simulated annealing, genetic algorithms, and tabu search. The genetic algorithm section is fairly standard, and contains much the same material as earlier papers. The section is short and does not give any specifics about the algorithm used. The final section in the paper discusses the results. It is fairly short, and does not give many details.

4.4.1 Comments

Overall, this paper is too short (5 pages) to be useful. It is well written, but lacks details. The tiny genetic algorithm section and few reported results make it difficult to re-implement or even to compare against. This paper is not a good candidate for further use.

- Typical results:
  - Convergence rates and computation time are the only reported results
- Re-implementation approach:
  - Parameters were not reported
  - Standard parameter values used in other re-implementations were used
  - Only the permutation attack was re-implemented as a later work from this author was used for the substitution attack
- Observed cryptanalytic usefulness:
  - Medium - the correct key was found with a high success rate
- Comment on result discrepancy:
  - No real discrepancy, different combination of metrics used

4.5 Vernam: Lin, Kao - 1995

This is the only paper published in 1995. By Feng-Tse Lin and Cheng-Yan Kao, [18] is a ciphertext-only attack on a Vernam cipher. The paper begins by introducing cryptography and cryptanalysis, including attack and cryptosystem types, as well as the Vernam cipher itself. The Vernam cipher is a one-time pad style system. Let $M = m_1, m_2, \ldots, m_n$ denote a plaintext bit stream and $K = k_1, k_2, \ldots, k_r$ a key bit stream. The Vernam cipher generates a ciphertext bit stream $C = E_k(M) = c_1, c_2, \ldots, c_n$, where $c_i = (m_i + k_i)$ mod p and p is a base. Since this is a one-key (symmetric or private-key) cryptosystem, the decryption formula has the same key bit stream K, only $M = D_k(C) = m_1, m_2, \ldots$, where $m_i = (c_i - k_i)$ mod p. The proposed approach is to determine K from an intercepted ciphertext C, and use it to break the cipher.

Since the described cryptosystem is not the Vernam cipher, and there are many errors in this paper, both in notation and logic, this application needs no further discussion. The genetic algorithm approach is rendered invalid by the errors present in the reported approach and assumptions.

4.5.1 Comments

This paper is of below average quality. It is rather short, and was written for the IEEE International Conference on Systems, Man, and Cybernetics. There are some problems with English grammar, possibly due to translation from the original language. The systems are adequately described, but are, however incorrect. The cipher given in this paper is not the Vernam cipher. A re-implementation is impossible due to the many errors present in this work.

- Typical results:
  - One run of the genetic algorithm is shown, with parameters
- Re-implementation approach:
  - Not re-implemented
- Observed cryptanalytic usefulness:
  - None - inconsistent notation, leading one to believe that the cipher attacked is not the true Vernam cipher

- Comment on result discrepancy:
  - No comment, not re-implemented

4.6 Merkle-Hellman Knapsack: Clark, Dawson, Bergen - 1996

This paper, by Clark, Dawson, and Bergen [7] is an extension of [25]. It contains a detailed analysis of the fitness function used in [25], as well as a modified version of the same fitness function. The paper begins by introducing the subset sum problem and previous work in genetic algorithm cryptanalysis. The fitness function from [25] is then discussed. Since the reason this function penalizes solutions which have a sum greater than the target sum for no clear reason, the function is altered to:

$$Fitness = 1 - (|Sum - Target|/MaxDiff)^{\frac{1}{2}} \tag{4.6.1}$$

The altered fitness function is found to be more efficient by running a genetic algorithm attack on 100 different knapsacks sums from the knapsack of size 15 used by Spillman [25]. This function requires fewer generations, on average, and searches less of the key space to find a solution.

The genetic algorithm attack is then introduced and Spillman's [25] attack summarized. In this work, for each of three different knapsack sizes - 15, 20, and 25, 100 different knapsack sums were formed, and the algorithm run until the solution was found. The amount of key space searched is about half that of an exhaustive attack, which is not a significant gain for a large knapsack. The gain is insignificant due to the overhead of the genetic algorithm approach. Also, the number of generations required, and the percentage of the key space searched vary widely between runs.

This attack is poor because the fitness function does not accurately describe the suitability of a given solution. The Hamming distance between the proposed solution and the true solution is not always indicated correctly by the fitness function. A high fitness value does not necessarily mean a low Hamming distance. The distribution of Hamming distances for a given range of fitness values appears to be binomial-like. Therefore, a fitness function based on a difference of sums is not adequate when attempting to solve the subset-sum problem with an optimization algorithm. Experimental results show that the genetic algorithm gives little improvement over an exhaustive search in terms of solution space covered, and, in fact, an exhaustive attack will be faster, on average, due to reduced algorithmic complexity.

4.6.1 Comments

This paper clearly discusses the problems with an earlier work, [25]. It is well written and gives a reasonable number of details. It is not, however a stand-alone work. A re-implementation is possible but unnecessary since the work establishes that the fitness function used is inappropriate. A comparison may be possible but unlikely since a new fitness function for the attack is needed.

- Typical results:
  - Reports the number of generations and the key space searched
  - These results are for three knapsack sizes (15, 20, 25), each attacked until the solution was found
- Re-implementation approach:
  - Not re-implemented
- Observed cryptanalytic usefulness:
  - None - this work disproves the usefulness of an earlier work [25]
- Comment on result discrepancy:
  - No comment, not re-implemented

4.7 Vigenère: Clark, Dawson, Nieuwland - 1996

This paper, by Clark, Dawson, and Nieuwland [8], is the first to use a parallel genetic algorithm for cryptanalysis. It attacks a polyalphabetic substitution cipher. The first section introduces the simple and polyalphabetic substitution ciphers, while the second section introduces the genetic algorithm. The parallel approach chosen for this work is to have a number of serial genetic algorithms working on a separate part of the problem. In this case, since more than one key must be found, each processor will work on finding a different key. Next, the application of the parallel genetic algorithm to the polyalphabetic substitution cipher is discussed. The fitness function uses weighted unigram, digram, and trigram frequencies as follows:

$$F(K) = w_1 \sum_{i=1}^{N} (K_1[i] - D_1[i])^2 + w_2 \sum_{i,j=1}^{N} (K_2[i,j] - D_2[i,j])^2 + w_3 \sum_{i,j,k=1}^{N} (K_3[i,j,k] - D_3[i,j,k])^2 \quad (4.7.1)$$

Here, K is a single N character key for a simple substitution cipher. K_1, K_2, and K_3 are the known unigram, digram, and trigram statistics, respectively. D_1, D_2, and D_3 are the statistics for the decrypted message, and w_1, w_2, and w_3 are weights chosen to equal one and emphasize different statistics. Since computation of a fitness function including trigrams can be computationally intensive, this work starts out using only unigrams and digrams, and later adds in the trigram statistics.

M processors in parallel are used, where M is the period of the polyalphabetic substitution cipher. Each processor works on one of the M keys, and communicates with other processors after a number of iterations. This communication involves sending the best key to each of a processor's neighbors, which allows digram and trigram statistics to be computed. The mating function is borrowed from [26] and requires the same special ordering of the key. Mutation has two possibilities, depending on the relative fitness of the child key. If the child has a fitness greater than the median in the gene pool, each character is swapped with the character to its right. If the child's fitness is less than the median, each character is swapped with a randomly chosen character in the key. The selection mechanism used is to combine all the children and all the keys from the previous generation together and select the most fit to be the new generation.

Fixed parameters for the experiment include:

- A gene pool size of 40
- A period of 3
- 200 iterations of the genetic algorithm
- A mutation rate of 0.05
- 10 iterations between slave-to-slave communications

The first test was done to determine the amount of known ciphertext needed to retrieve a significant portion of the plaintext. A plot was created, with each point found by running the algorithm on ten different encrypted messages, five times per message. In the five runs per message, the random number generator was seeded differently every time. About 90% of the original message was recovered using 600 known ciphertext characters per key. In this test, the weights were $w_1 = 0.4, w_2 = 0.6$, and $w_3 = 0.0$ initially, and $w_1 = 0.2, w_2 = 0.3$, and $w_3 = 0.5$ after 100 iterations.

The second test was done to indicate the optimal choice of weights. The weights were chosen such that

$$w_1, w_2, w_3 \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$$

and $\sum_{i=1}^{3} w_i = 1.0$. There are 66 combinations that satisfy these conditions. The algorithm was run 20 times per combination on ten different messages, twice per message. Of the two runs per message, the best result was taken, with the ten best results then averaged together. The values of w_1, w_2 and w_3 were fixed for all 200 iterations of the algorithm. The best results occurred when $w_3 = 0.8$.

4.7.1 Comments

This paper is short but reasonably detailed. The algorithms are clearly defined and the result graphs are a nice touch. However, only the parallel case is considered, making the paper of limited usefulness. Re-implementation and comparison are reasonably possible, both in a serial and a parallel version. If re-implemented, the serial version would be created first to determine if parallelization is warranted.

- Typical results:
  - Reports on the amount of known ciphertext required in order to regain a certain percentage of the original message
  - Reports on the optimal choice of fitness function weights
- Re-implementation approach:
  - Re-implemented in a serial version only
  - Most of the original parameters were used in the re-implementation
  - Additional parameters are used to expand the range of parameter values
- Observed cryptanalytic usefulness:
  - None - no messages were successfully decrypted
- Comment on result discrepancy:
  - Different conclusions based on the different metrics used

4.8 Vigenère: Clark, Dawson - 1997

This is the first paper published in 1997. By Clark and Dawson, [5] is, overall, a slightly more detailed, longer version of [8]. Therefore, it shares all the same characteristics and is evaluated as above.

4.9 Merkle-Hellman Knapsack: Kolodziejczyk - 1997

This paper [17], published in 1997, is by Kolodziejczyk. It is an extension of [25]. It focuses on the Merkle-Hellman knapsack system, and the effect of initial parameters on the approach reported in [25]. The paper introduces the Merkle-Hellman knapsack, and then discusses the application of a genetic algorithm to the cryptanalysis thereof. Certain restrictions are placed on the encoding algorithm. These are:

- Only the ASCII code will be used in encryption
- The superincreasing sequence will have 8 elements, ensuring that each character has a unique encoding
- The plaintext is not more than 100 characters in length

Due to these restrictions, the knapsack is a completely trivial problem. The genetic algorithm approach needs no discussion, as this application has no point.

4.9.1 Comments

Overall, this paper presents a reasonable amount of numerical data and is a good extension work to [25]. Unfortunately, the paper is not well written, has poor English grammar and word usage, and uses a completely trivial knapsack. The only contribution of this paper is the addition of parameter variation to [25]. There is no point in reimplementation of this paper.

- Typical results:
  - Used 5 ASCII characters, each of which was decrypted
- Re-implementation approach:
  - Not re-implemented

- Observed cryptanalytic usefulness:
    - None - knapsack size is trivial
- Comment on result discrepancy:
    - No comment, not re-implemented

4.10 Substitution: Clark, Dawson - 1998

This is the only paper published in 1998. By Clark and Dawson, [6] compares three optimization algorithms applied to the cryptanalysis of a simple substitution cipher. These algorithms are simulated annealing, the genetic algorithm, and tabu search. Performance criteria, such as speed and efficiency, are investigated. The paper begins by introducing the application area and giving the basis for the current work. A simple substitution cipher is attacked by all three algorithms, with a new attack for tabu search, as compared to the attack done in [4]. Each of the attacks are compared on three criteria:

- The amount of known ciphertext available to the attack
- The number of keys considered before the correct solution was found
- The time required by the attack to determine the correct solution

The next section of the paper gives a detailed overview of each of the three algorithms. It is followed by an overview of the attack, as well as details about the attack for each algorithm. The overall attack uses a general suitability assessment or fitness function. Assuming $\mathcal{A}$ denotes the language alphabet, here $\{A, \ldots, Z, \_\}$ (where $\_$ is the symbol for a space), the function is:

$$C_k = \alpha \sum_{i \in \mathcal{A}} |K^u_{(i)} - D^u_{(i)}| + \beta \sum_{i,j \in \mathcal{A}} |K^b_{(i,j)} - D^b_{(i,j)}| + \gamma \sum_{i,j,k \in \mathcal{A}} |K^t_{(i,j,k)} - D^t_{(i,j,k)}| \quad (4.10.1)$$

k is the proposed key, K is the known language statistics, D is the decrypted message statistics, and $u/b/t$ are the unigram, digram, and trigram statistics. $\alpha, \beta,$ and γ are the weights. The complexity of this formula is $\mathcal{O}(N^3)$, where N is the alphabet size, when trigram statistics are used. The effectiveness of using the different n-grams is evaluated using a range of weights, and it is concluded that trigrams are the most effective basis for a cost function, but the benefit of using trigrams over digrams is small. In fact, due to the added complexity of using trigrams, it is usually more practical to use only unigrams and digrams. Therefore, the remainder of the paper uses a fitness function based only on digrams. This function is:

$$C_k = \sum_{i,j \in \mathcal{A}} |K^b_{(i,j)} - D^b_{(i,j)}| \tag{4.10.2}$$

The genetic algorithm attack section begins with detailed pseudocode for the mating process, which is similar to that used in [26]. Selection is random, with a bias towards the more fit chromosomes. Next, the mutation operator is discussed. Mutation randomly swaps two characters in a chromosome. Pseudocode is then given for the overall genetic algorithm attack. The attack parameters are M, the solution pool size, and MAX_ITER, the maximum number of iterations. The M best solutions after reproduction continue on to the next iteration.

Experimental results for the three algorithms were generated with 300 runs per data point. One hundred different messages were created for each case, and the attack run three times per message. The best result for each message was then averaged together to produce the data point. This is done to provide a good representation of the algorithm's ability. In overall outcome of the attack, all the algorithms performed approximately the same. The algorithms also perform equally well when comparing the amount of ciphertext provided in the attack. Considering complexity, however, simulated annealing is more efficient than the genetic algorithm, and tabu search is the most efficient.

4.10.1 Comments

Overall, this paper is well written and provides a good level of detail. It provides good discussion of important points, especially in interpretation of the results. It is not focused on a genetic algorithm attack, and only uses a simple substitution cipher, however, the details given make a re-implementation fairly easy, and provide enough information for comparison purposes. This paper is a good choice for further study.

- Typical results:
  - Results include number of key elements correct vs. amount of known ciphertext, number of key elements correct vs. total keys considered and number of key elements correct vs. time
- Re-implementation approach:
  - Parameters were not reported
  - Standard parameter values used in other re-implementations were used
- Observed cryptanalytic usefulness:
  - None - no successful decryptions of the message occurred
- Comment on result discrepancy:
  - Different conclusions based on the different metrics used

4.11 Chor-Rivest Knapsack: Yaseen, Sahasrabuddhe - 1999

This is the only paper from 1999. By Yaseen and Sahasrabuddhe, [31] is also the only paper that considers a genetic algorithm attack on the Chor-Rivest public key cryptosystem. The paper begins with a review of the Chor-Rivest scheme, followed by a review of genetic algorithms. The genetic algorithm attack is then discussed.

The input to the genetic algorithm is $A = (a_1, \ldots, a_n)$, a knapsack of positive integers, and c, an integer which represents the cipher. The output is $M = (m_1, \ldots, m_n)$, a binary vector such that $c = A \cdot M$. A and M are both binary vectors. The binary vector A is the chromosome that represents the solution message M. The length of the vector is

n, the same as the size of the knapsack. $a_i = 1$ if the element has been chosen and 0 otherwise, the same as in [25]. The initial population is created randomly, such that each n-bit binary chromosome contains exactly h 1's, where h is the degree of the monic irreducible polynomial $f(x)$ over Z_p. The fitness function is:

- Determine the value of the current chromosome (call it Sum)
- $$Fit = 1 - \sqrt{|Sum - Target|/FullSum} \tag{4.11.1}$$

  where $FullSum$ is the sum of all the components in the knapsack

Two genetic operators are used during the alteration (mutation) phase of the algorithm. They are:

- Swapbits: choose two positions in the chromosome randomly and swap their contents
- Invert: choose two positions in the chromosome randomly and reverse the sequence of bits between them

The parameters used throughout the experiment are:

- p_{inv} (the probability of the invert operation occurring) = 0.75
- p_m (the probability of the swapbits operation occurring) initially = 0.75
- Population size of 30

To assist the fitness function, the chromosomes with a Hamming distance of 2 and 4 from a target were the ones used. This approach is based on some observations made in [7]. Experimental results were provided for the knapsacks of the fields $GF(13^5), GF(17^5), GF(19^5)$, and $GF(23^5)$. These results are for the average of 30 experiments, both with and without the Hamming distance technique. With the Hamming technique, as the size of the knapsack increased, the percentage of search space visited decreased. Without the technique, the percentage visited had no relation to the knapsack size. With the technique, 1 to 2 % of search space was visited, versus 50 to 90 % without. The average number of generations showed a similar increase - 1 to 10 generations with the technique and 23 to 900 generations without, on average. Of the 120 experiments done in each case, 33 experiments without the technique did not reach a solution even after 5000 generations.

4.11.1 Comments

The main focus of this paper is on the Hamming distance technique, which is not explained that clearly. A reasonable level of detail is included, however, due to length (5 pages) and clarity issues, this paper is a poor choice for re-implementation.

- Typical results:
  - Reports the average number of chromosomes required to find an exact solution
- Re-implementation approach:
  - Not re-implemented
- Observed cryptanalytic usefulness:
  - None - lacks information about parameters required to analyze the attack
- Comment on result discrepancy:
  - No comment, not re-implemented

4.12 Substitution/Transposition: Gründlingh, Van Vuuren - submitted 2002

This paper has not yet been published, but was submitted for publication in 2002. By Gründlingh and Van Vuuren, [13] combines operations research with cryptology and attacks two classical ciphers with a genetic algorithm approach. The two ciphers studied are substitution and transposition. The paper begins with some historical background on the two fields, and then covers basic components of symmetric (private key) ciphers. The next two sections discuss genetic algorithms and letter frequencies in natural languages. The main difficulty in a genetic algorithm implementation is said to be the consideration of constraints. As for character frequencies, these were generated for 7 languages from modern bible texts.

The mono-alphabetic substitution cipher was attacked first. In this case, consider a ciphertext T of length N, formed from a plaintext of the same length in a natural language L. The fitness function of a candidate key $\underline{k}$, is:

$$\mathcal{F}_1(L,N,T,\underline{k}) = \frac{2(N-\rho^N_{min}(L)) - \sum_{i=A}^{Z}|\rho^N_i(L) - f_i^{T(N)}(\underline{k})|}{2(N-\rho^N_{min}(L))} \tag{4.12.1}$$

Here, $\rho_i^N(L)$ represents the (scaled) expected number of times the letter i will appear per N characters in a natural language L text. $f_i^{T(N)}(\underline{k})$ is the observed number of occurrences of letter i when the ciphertext T, of length N, is decrypted using the key $\underline{k}$. Also, $\rho^N_{min}(L) = min_i\{\rho^N_i(L)\}$. The limits of $\mathcal{F}_1$ are $[0,1]$, and both bounds may be reached in extreme cases. The rationale behind this fitness function involves discrepancies between the expected number of occurrences and the observed number of occurrences of a letter i. An alternative option for the fitness measure, this time including digrams is:

$$\begin{aligned}\mathcal{F}_2(L,N,T,\underline{k}) = [2(2N-1-\rho^N_{min}(L)-\delta^N_{min}(L)) - \sum_{i=A}^{Z}|\rho^N_i(L) - f_i^{T(N)}(\underline{k})| \\ - \sum_{i,j=A}^{Z}|\delta^N_{ij}(L) - f_{ij}^{T(N)}(\underline{k})|] \div (2(2N-1-\rho^N_{min}(L)-\delta^N_{min}(L)))\end{aligned} \tag{4.12.2}$$

Here, $\rho_i^N(L)$ and $\delta_{ij}^N(L)$ represent the expected number of occurrences of, respectively, the letter i and the digram ij, per N characters in a natural language L text. $f_i^{T(N)}(\underline{k})$ and $f_{ij}^{T(N)}(\underline{k})$ are the observed number of occurrences of, respectively, the letter i and the digram ij, when the ciphertext T is decrypted using the key $\underline{k}$. Also, $\rho^N_{min}(L) = min_i\{\rho^N_i(L)\}$ and $\delta^N_{min}(L) = min_{ij}\{\delta^N_{ij}(L)\}$. The same limits apply to $\mathcal{F}_2$ as to $\mathcal{F}_1$.

A detailed algorithm overview, including pseudocode, is discussed next, including the genetic operators of mutation, crossover, and selection. The mutation operator is a single random swap of two characters. Crossover occurs during a traversal of the parent chromosomes. The first child chromosome is produced as follows:

- Begin a left-to-right traversal of the parent chromosomes
- Select character i from the parent whose i-th character has the better frequency match to the expected number of occurrences of i
- If the i-th character already appears in the child
  - Select the character from the other parent
  - If the characters are both already in the child, leave the position blank
- If any characters do not appear in the child, insert them to minimize discrepancies between observed and expected frequency counts

The second child is produced by the same process using a right-to-left traversal. Keys are selected stochastically, such that keys that have a higher fitness value are more likely to be mated. The algorithm was run on a ciphertext of 2,519 characters and a population size of 20. Using (4.12.1) an average pool fitness of 90 % was reached within 7 generations, and using (4.12.2), an average of 75 % within 15 generations. The algorithm was run for 200 generations, and sets of the best 3 keys overall were kept as well as the final population.

The transposition cipher was attacked second, using a similar genetic algorithm approach. This attack assumes that the number of columns is known, but the column shuffling is unknown. The fitness function then becomes (using the same symbols as before):

$$\mathcal{F}_3(L,N,T,\underline{k}) = \frac{2(N-1-\delta^N_{min}(L)) - \sum_{i,j=A}^{Z}|\delta^N_{ij}(L) - f^{T(N)}_{ij}(\underline{k})|}{2(N-1-\delta^N_{min}(L))} \tag{4.12.3}$$

The crossover operator uses a random binary vector to select entries from the parents. The first child is formed by taking values from parent 1 where 1 occurs in the binary vector. The rest of the values are filled from parent 2, in the order in which they occur. The second child is formed starting from parent 2, and using the locations of the 0's in the binary vector. Two mutation operators are considered - point and shift. A point mutation consists of randomly selecting two positions in a key and swapping their values. Shift mutation, on the other hand, consists of shifting the values in a key a random number of positions to the left or right modulo the length of the key. Otherwise, the same algorithm was used in this attack as for the simple substitution cipher. No convergence of fitness values was found for this attack.

The paper concludes by describing a decision tool created to assist cryptanalysts who routinely attack substitution ciphers. This tool is graphical in nature, and is based on the genetic algorithm attack on substitution ciphers discussed above.

4.12.1 Comments

Overall, this paper is pretty good and includes good introductory material. It lacks enough detail in the results section to be easily comparable, but gives a great level of detail for the functions, algorithms, and operators used. It is a good candidate for re-implementation, but not for comparison.

- Typical results:
  - Gives the final population pool and the partially decrypted ciphertext produced by the best key for the substitution attack
  - Transposition attack declared to be nonconvergent and no results given
- Re-implementation approach:
  - Both the substitution and transposition ciphers were re-implemented
  - Different text lengths used, same parameters re-used
  - Additional parameters are used to expand the range of parameter values
- Observed cryptanalytic usefulness:
  - Substitution: None - no messages successfully decrypted

    - Transposition: Low - low decryption percentage

- Comment on result discrepancy:

    - Different conclusions based on the different metrics used

4.13 Overall Comments

The quality of the reference works found in the literature varies widely. This issue is compounded by the low number of reference works found - twelve. Several of these works were too poorly specified for further use or used trivial parameters, leaving even fewer valid reference works. Seven works contained enough information for a re-implementation to be completed. Some of these works depended on other, earlier works for the re-implementation to be possible. A common set of metrics for these re-implemented works is gathered. It is not possible to compare the results using the new metrics to the results reported in the previous works due to the differences in metric composition. The metrics previously used are a diverse collection, with very different bases. This diversity is one of the motivating factors for this work.

Chapter 5

Traditional Cryptanalysis Methods

Only four of the seven ciphers attacked using a genetic algorithm will be further investigated. These ciphers are monoalphabetic substitution, polyalphabetic substitution (e.g. the Vigenère cipher), permutation and columnar transposition. The other three ciphers are excluded due to various reasons.

The Merkle-Hellman knapsack cipher was initially marked for further study but has been removed due to the difficulty of implementing the seminal attack by Shamir [23]. This attack involves Lenstra's integer programming algorithm and a fast diophantine approximation algorithm. The Chor-Rivest knapsack cipher is difficult to implement and was only considered for a genetic algorithm attack in one paper, which used trivial examples of the system. The paper [31] is short and does not explain the attack technique well, making it difficult to re-implement the work. The third cipher that will not be considered further is the Vernam cipher. This cipher is one of the few that exhibits perfect secrecy, however, it is not widely used due to the key length requirements. This cipher is also attacked using a genetic algorithm in only one paper [18] which really is not attacking the cipher it claims it is. Therefore, due to lack of replicable works, this cipher is excluded from further investigation.

5.1 Distinguishing among the three ciphers

If the cipher type is not already known, inspection of the frequency of occurrence of the ciphertext letters (i.e. how often each ciphertext character appears in the ciphertext), will distinguish between a permutation and a substitution cipher. Each language (e.g. French, English, German) has a different set of frequencies for its letters. A permutation cipher will produce a ciphertext with the same frequency distribution as that of the language of the plaintext [24]. For example, if the plaintext was written in English, a permutation cipher will produce ciphertext with the same frequency distribution as any similar English text.

The index of coincidence (IC) measures the variation in letter frequencies in a ciphertext [22]. If simple (monoalphabetic) substitution has been used, then there will be considerable variation in the letter frequencies and the IC will be high [22]. As the period of the cipher increases (i.e. more alphabets are used), the variation in letter frequencies decreases and the IC is low.

5.2 Monoalphabetic substitution cipher

The traditional attack for the monoalphabetic substitution cipher utilizes frequency analysis and examination. The first step is to analyze the ciphertext to determine the frequency of occurrence of each character [27]. The frequencies of digrams and trigrams may also be determined in this step. Then, based on the frequencies, the cryptanalyst examines the ciphertext and proceeds by guessing and changing guesses for which cipher character is which plaintext character.

For example, the most frequent character in an English text is E [27]. Therefore, the ciphertext character that appears most frequently is highly likely to be the equivalent of the plaintext E. The next most frequent English letters are T, A, O, I, N, S, H, and R, all with around the same frequency. Digram and trigram statistics can be used to help identify these. TH, HE, and IN are the three most common digrams, and the three most common trigram are THE, ING, and AND. In the end, however, the cryptanalyst must gain a feel for the text and proceed by guessing letters until the text becomes legible.

5.3 Polyalphabetic substitution cipher

The polyalphabetic substitution cipher (here, the Vigenère cipher) is attacked using two techniques - the Kasiski test and the index of coincidence [27]. The first step is to determine the keyword length, denoted m.

The Kasiski test was described by Friedrich Kasiski in 1863, but apparently discovered earlier, around 1854, by Charles Babbage [27]. This test is based on the observation that "two identical segments of plaintext will be encrypted to the same ciphertext whenever their occurrence in the plaintext is δ positions apart, where $\delta \equiv 0 \pmod{m}$" [27]. Conversely, if two segments of ciphertext are identical and have at least a length of three, then there is a good chance they correspond to identical segments of plaintext [27].

The Kasiski test follows the below procedure [27]:

1. Search the ciphertext for pairs of identical segments of length at least three
2. Record the distance between the starting positions of the two segments
3. If several such distances are obtained ($\delta_1, \delta_2, \ldots$),
    - m divides all of the δ_i's
    - and, therefore, m divides the greatest common divisor (gcd) of the δ_i's

The index of coincidence is used to further verify the value of m. This concept was defined by Wolfe Friedman in 1920, as follows [27]:

- Suppose $x = x_1 x_2 \cdots x_n$ is a string of n alphabetic characters
- The index of coincidence of x, denoted $I_c(x)$ is defined to be the probability that two random elements of x are identical

Suppose the frequencies of $A, B, C, \ldots, Z$ in x are denoted by $f_0, f_1, \ldots, f_{25}$, respectively. Two elements of x can be chosen in $\binom{n}{2}$ ways [27]. For each i, $0 \leq i \leq 25$, there are $\binom{f_i}{2}$ ways of choosing both elements to be i. This produces the formula [27]:

$$I_c(x) = \frac{\sum_{i=0}^{25} \binom{f_i}{2}}{\binom{n}{2}} = \frac{\sum_{i=0}^{25} f_i(f_i - 1)}{n(n-1)}$$

Suppose x is a string of English text. Then, if the expected probabilities of occurrence of the letters $A, B, \ldots, Z$ are denoted by $p_0, \ldots, p_{25}$, respectively, we would expect that $I_c(x) \approx \sum_{i=0}^{25} p_i^2 = 0.065$ [27]. The same reasoning applies if x is a ciphertext string obtained using any monoalphabetic cipher - the individual probabilities are permuted but the quantity $\sum p_i^2$ is unchanged.

Now, suppose we have a ciphertext string $\mathbf{y} = y_1 y_2 \cdots y_n$ produced by a Vigenère cipher. Define m substrings of $\mathbf{y}$, denoted $\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_m}$, by writing out the ciphertext, in columns, in a rectangular array of dimensions $m \times (n/m)$ [27]. The rows of this matrix are the substrings $\mathbf{y_i}$, $1 \leq i \leq m$. This means that we have:

$$\begin{aligned}
\mathbf{y_1} &= y_1 y_{m+} y_1 y_{2m+} \cdots \\
\mathbf{y_2} &= y_2 y_{m+} y_2 y_{2m+} \cdots \\
\vdots & \;\vdots\; \vdots \\
\mathbf{y_m} &= y_m y_{2m} y_{3m} \cdots
\end{aligned}$$

If $\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_m}$ are constructed as shown, and m is indeed the keyword length, then $I_c(\mathbf{y_i})$ should be approximately equal to 0.065 for $1 \leq i \leq m$. If m is not the keyword length, then the substrings $\mathbf{y_i}$ will look much more random since they will have been produced by shift encryption with different keys. A completely random string produces an $I_c \approx 26(\frac{1}{26})^2 = \frac{1}{26} = 0.038$ [27]. These two values (0.065 and 0.038) are far enough apart that the correct keyword length can be obtained or confirmed using the index of coincidence.

After the value of m has been determined, the actual key, $K = (k_1, k_2, \ldots, k_m)$ needs to be found. The method in [27] is as follows:

- Let $1 \leq i \leq m$ and $f_0, \ldots, f_{25}$ denote the frequencies of $A, B, \ldots, Z$ in the string $\mathbf{y_i}$
- Let $n' = n/m$ be the length of the string $\mathbf{y_i}$
- The probability distribution of the 26 letters in $\mathbf{y_i}$ is then $\dfrac{f_0}{n'}, \ldots, \dfrac{f_{25}}{n'}$
- Since each substring $\mathbf{y_i}$ is obtained by shift encryption of a subset of the plaintext elements using a shift k_i, it is hoped that the shifted probability distribution $\dfrac{f_{k_i}}{n'}, \ldots, \dfrac{f_{25+k_i}}{n'}$ will be close to the ideal probability distribution $p_0, \ldots, p_{25}$, with subscripts evaluated modulo 26
- Suppose that $0 \leq g \leq 25$ and define the quantity $M_g = \displaystyle\sum_{i=0}^{25} \frac{p_i f_{i+g}}{n'}$
- If $g = k_i$, then $M_g \approx \sum_{i=0}^{25} p_i^2 = 0.065$
- If $g \neq k_i$, then M_g will usually be significantly smaller than 0.065

This technique should allow the determination of the correct value of k_i for each value of $i, 1 \leq i \leq m$.

5.4 Permutation/Transposition ciphers

The permutation cipher defined in [27] is the same as that used in [4], while [20] and [13] use a columnar transposition cipher. The permutation cipher will be attacked by exhaustion as per [22]. This attack consists of considering $d!$ permutations of the d characters, where d is the period of the cipher. The ciphertext is divided into blocks of various lengths, and all possible permutations considered. For each possible value of d, starting at $d = 2$, all $d!$ possible permutations are attempted on the first two groups of d letters. For example, if $d = 2$, the first two groups of 2 letters are permuted as (letter 1)(letter 2) and (letter 2)(letter

1). This continues until one or both of the groups permute into something that looks like a valid word or word part. At this time, the possibly valid permutation is applied to the other groups of letters in the ciphertext. If the permutation produces valid text, i.e., the human cryptanalyst accepts the text as valid in the language used, then the attack is completed, otherwise, the process continues.

The columnar transposition cipher is attacked differently for an incompletely filled rectangle than for a completely filled rectangle. We will assume that the rectangle is completely filled in order to simplify the attack. The width of a completely filled rectangle must be a divisor of the length of the message, i.e., if the message has 77 letters, the rectangle must be either 7 or 11 letters wide [24]. The ciphertext is written vertically into rectangles of the possible widths, and a solution obtained by rearranging the columns to form plaintext. The process of restoring a disarranged set of letters into its original positions is called anagramming [24].

There are several ways to proceed with the process of anagramming. These include the use of a probable word, the use of a probable sequence, digram/trigram charts, contact charts, and vowel frequencies. A contact chart is an extension of the digram/trigram charts, and contains information relating to the likelihood that a specific letter will be preceded or followed by another letter. Vowel frequencies are language-specific. In English, vowels comprise approximately 40% of the text, independent of text length [11]. Therefore, when putting the ciphertext into matrix (rectangle) form, a period should be chosen which produces the most even distribution of vowels across the columns of the matrix [19]. The text is deciphered more easily if the columns are then exchanged so that the most frequently occurring letter combinations appear first [19]. This fact helps the cryptanalyst to determine the most likely possible width for the inscription rectangle.

The next step after vowel frequency examination is usually the use of a probable word or sequence. The probable word is used when the cryptanalyst has some information about the message which would lead to the assumption of a word in the text [24]. For example, if the message is of a military nature, words such as division, regiment, attack, report, communication, enemy, retreat, etc. are likely to be present [11]. If a probable word can not be assumed, the next option is to look for the presence of one of the less frequent digrams such as QU or YP [11]. If this is not possible either, reconstruction may default to working on the first word of the message as in [24].

If there is some difficulty involved in anagramming the first row, the next step is to try to fit two columns together to produce good digraphs [24]. One technique for column fitting is to give each assumed digram its frequency in plaintext and then add the frequencies [15]. The combination with the highest total is most likely to be correct [15]. Next, one would attempt to extend the digrams into trigrams, and so on. Columns can be added both before and after the current set, so the extension into trigrams and longer combinations must consider both situations.

5.5 Comments

Overall, the traditional attack methods are not easily automated, and tend to require a trained and experienced cryptanalyst. The polyalphabetic substitution cipher is the only one that is attacked mostly mathematically. Mathematical information is used for both the monoalphabetic substitution cipher and the permutation/transposition ciphers, however, the main component of these attacks is the human cryptanalyst. The final decision on whether or not the plaintext is found is always made by the human cryptanalyst, no matter which method is used in the attack.

Chapter 6

Attack Results

6.1 Methodology

The main metric for both classical and genetic algorithm attacks was elapsed time. This was measured using the tcsh built-in command, “time”. In the classical algorithm case, the elapsed time is measured from the time the attack begins to the time the ciphertext is completely decrypted. This measurement is almost completely comprised of time spent interacting with the user, as processing time is less than is measurable with the “time” command. In the genetic algorithm case, each attack was run using a tcsh shell script. Therefore, the amount of time spent in user interaction is minimal, and the elapsed time measurement is comprised almost wholly of processing time. The differences in composition of elapsed time for the two categories of attack make it difficult to compare the attacks on a time basis. Therefore, an additional metric is used for the genetic algorithm attacks.

This additional metric, used only for the genetic algorithm attacks, is the percentage of attacks which completely decrypted the ciphertext, as determined by the human cryptanalyst upon examination of the final result. This metric is invalid in a classical attack, as this class of attacks run until the text is decrypted. A genetic algorithm attack, on the other hand, runs for a specific number of generations, independent of the decryption of the ciphertext. Using this metric, as well as the elapsed time, gives one a better feel for the usefulness of each attack. The time measurement is irrelevant if the attack is unsuccessful.

These two metrics were chosen so that traditional and genetic algorithm attacks could be compared. Elapsed time measures the efficiency of the attack while the percentage of successful attacks measures how useful the attack is. Both of these metrics are needed to gauge the success of an attack.

The same set of ciphertexts was used for all attacks. This set consists of 100, 500, and 1000 characters of five different English texts. These texts are: Chapter 1 of Genesis from the King James Version of the *Bible* [2], Patrick Henry's "Give me liberty or give me death!" speech [3], Section 1/Part 1 of Charles Darwin's *The Origin of Species* [10], Chapter 1 of H. G. Wells's *The Time Machine* [30], and Chapter 1/Part 1 of Louisa May Alcott's *Little Women* [1].

Additional parameters were required for the genetic algorithm attacks. The parameters were compiled from the various genetic algorithm attacks found in the literature and then extended to cover greater parameter space. All attacks were run using the following parameters:

- Population sizes of 10, 20, 40, or 50
- Number of generations equaling 25, 50, 100, 200 or 400
- Mutation rates of 0.05, 0.10, 0.20, 0.40

Each possible combination of parameters was run once per ciphertext and length combination.

Certain genetic algorithm attacks required additional parameters. The columnar transposition attacks were run using keylengths of 7, 9, or 11. The Vigenère attack was run using a period of 3, 5, or 7. The fitness weights used in the Vigenère attack were: 0.1 for unigrams and digrams, 0.8 for trigrams. The permutation attack was run using keylengths of 2, 4, or 6. The only attack that varied from others attacking its cipher type was the Matthews [20] columnar transposition attack. An additional set of parameters was used for this attack. These parameters were:

- Initial and Final Crossover probabilities

- Initial and Final Point Mutation probabilities
- Initial and Final Shuffle Mutation probabilities

In order to use the same set of parameters as the other attacks, both crossover probabilities were set to 1 and both of the point mutation and shuffle mutation values were set to 0.05, 0.10, 0.20 or 0.40. The additional set of parameters used consisted of setting the initial crossover probability to 0.8, the final crossover probability to 0.5, the initial point mutation probability to 0.1, the final point mutation probability to 0.5, the initial shuffle mutation probability to 0.1, and the final shuffle mutation probability to 0.8. This additional set of parameters was used throughout the original results reported for the Matthews attack.

6.2 Classical Attack Results

As stated above, all classical attacks are run until complete decryption of the ciphertext is obtained. Since each ciphertext is different in both statistics and distribution of common/uncommon words, the time needed to decrypt a text can vary widely. The more mathematical an attack is, the shorter the period of time required.

Since a classical substitution attack is based on statistical information about the text, as well as the user's intuition, the time this attack takes can vary widely from text to text. Usually, a longer text is easier to decrypt, as the statistical characteristics are more distinct. Some texts are statistically very well-defined, making them easy to decrypt even at shorter text lengths. The first chapter of Genesis, for example, is easy to decrypt due to the clarity of the statistical evidence. Other texts, that use less common words or word-patterns, such as the Liberty or Death speech, are consistently difficult.

As Figure 6.1 shows, times can vary widely between texts, and between different lengths of the same text. Times range from 1:50 to 10:25, and can vary as much as 6:25 or as little as :45 between different lengths of a text.

The classical Vigenère attack, on the other hand, is very mathematically based. This means that the time to decrypt a text is much more consistent. Longer decryption times

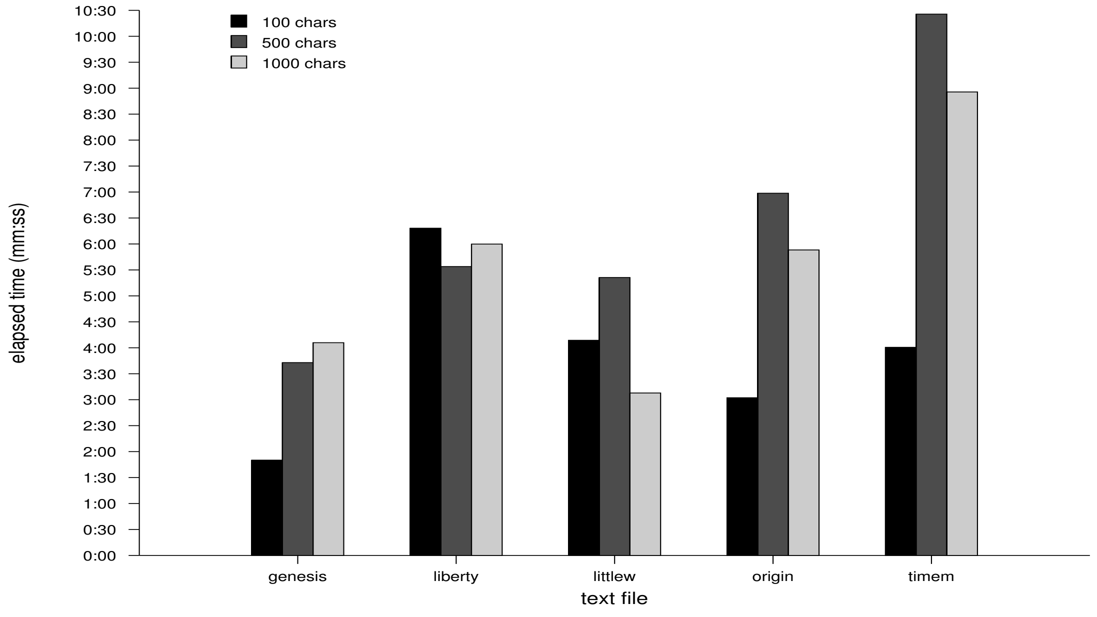

Figure 6.1: Classical Substitution Attack

mean that there was more than one good candidate for the period, leading to multiple attempts at decryption. This is what occurred for most of the 100 character texts when a period of 5 or 7 was used.

As Figure 6.2 shows, times are fairly consistent among different texts and text lengths. Times range from :58 to 6:25, and vary as much as 4:55 or as little as :10 between different text lengths.

The classical permutation attack is a brute force, exhaustive search. Therefore, the amount of time the attack takes depends both on the characteristics of the text and the block length used. A longer block length takes more time to decrypt because there are many more possibilities that must be considered and sorted through. Texts that contain less common character combinations will take longer to decrypt because fewer permutations result in that particular combination.

Figure 6.3 shows that this attack is more consistent for elapsed time values at smaller, more trivial block lengths. At longer block lengths, it takes more time to sort through the possible permutations to determine if a correct arrangement has been found. The word

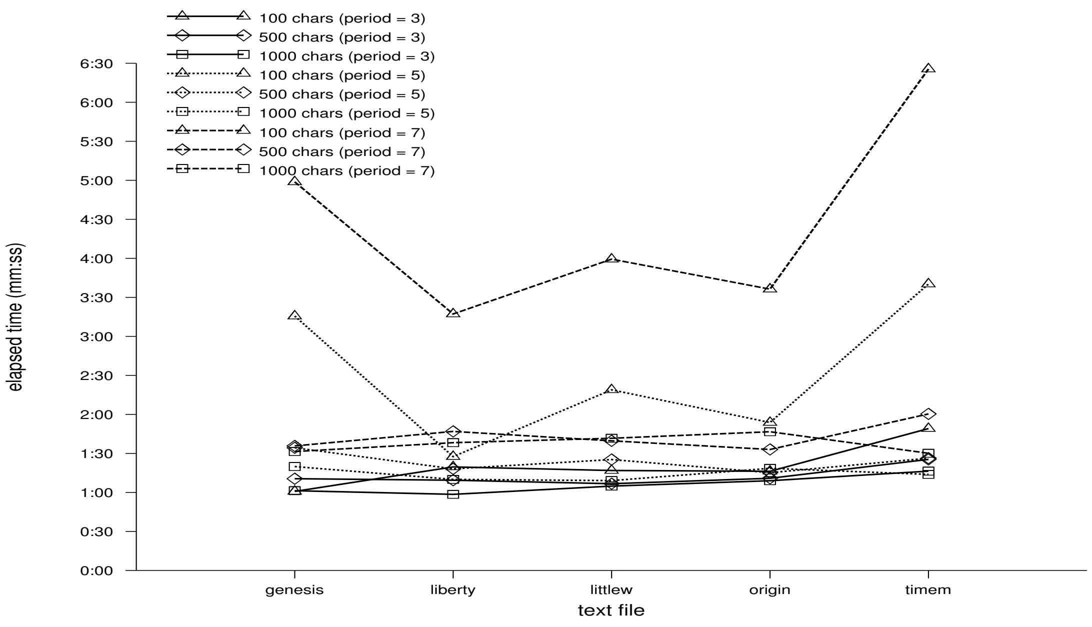

Figure 6.2: Classical Vigenère Attack

choice of the text can make it more difficult to determine if a correct permutation has been found, since longer or less common words are more difficult to distinguish from a partial root. Times range from :13 to :15 for texts at block length 2, :21 to :38 for texts at block length 4, and 1:21 to 2:25 for texts at block length 6.

A classical columnar transposition attack begins with statistical analysis of the ciphertext. The user then proceeds with anagramming, or rearranging the columns to produce recognizable words. Like the classical substitution attack, a great deal depends on the user's intuition. The more often easily recognizable words occur in the text, the easier and faster the user can rearrange the columns correctly. The characteristics of the text, therefore, are a significant factor in the length of time required to decrypt the text.

As shown in Figure 6.4, block length does not heavily influence the time it takes to decrypt a text. The text attacked, and its length, are more influential factors. Times range from :30 to 3:15, with certain texts fairly consistently taking more time.

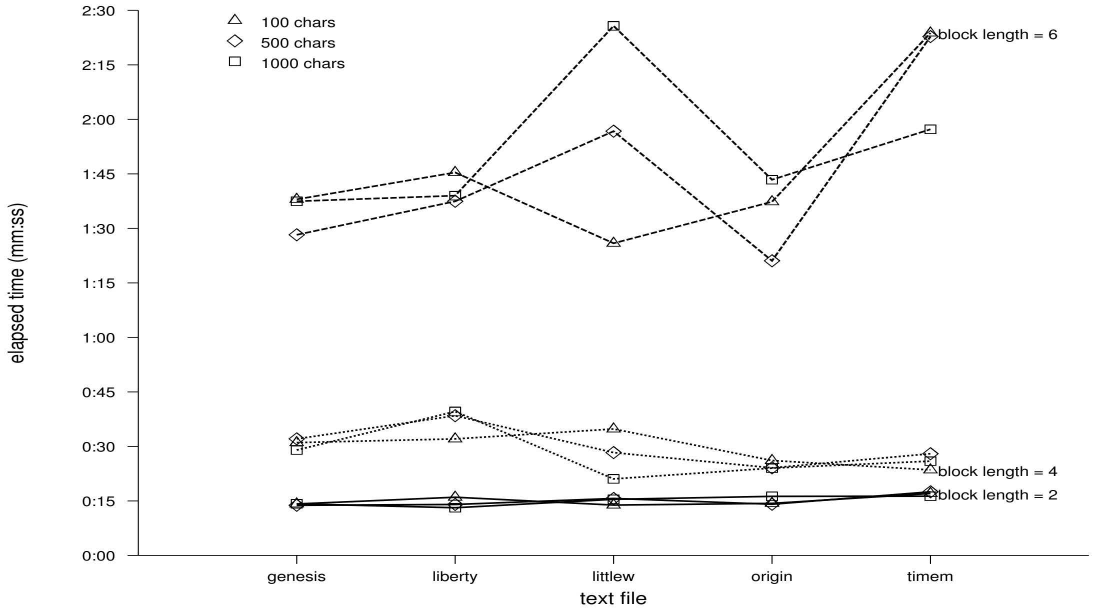

Figure 6.3: Classical Permutation Attack

6.3 Genetic Algorithm Attack Results

Only the attacks on the monoalphabetic substitution, polyalphabetic substitution (i.e. the Vigenère cipher), permutation and columnar transposition were selected for further investigation. This includes the attacks by Spillman et al. (1993) [26], Matthews (1993) [20], Clark (1994) [4], Clark et al. (1996)[8] / Clark & Dawson (1997) [5], Clark & Dawson (1998) [6], and both attacks from Gründlingh & Van Vuuren (2002) [13].

The majority of the Genetic Algorithm based attacks did not correctly decrypt any texts. Two of the attacks, Matthews (1993) [20] and Gründlingh & Van Vuuren (2002) [13], decrypted some texts correctly. One attack, Clark (1994) [4] correctly decrypted many texts. Only the three successful attacks will be discussed here, as time measurements for the four unsuccessful attacks are irrelevant.

The Matthews (1993) [20] attack and the Gründlingh & Van Vuuren (2002) [13] attack were against the columnar transposition cipher, while the Clark (1994) [4] attack was against the permutation cipher. The Matthews attack was successful only for key length

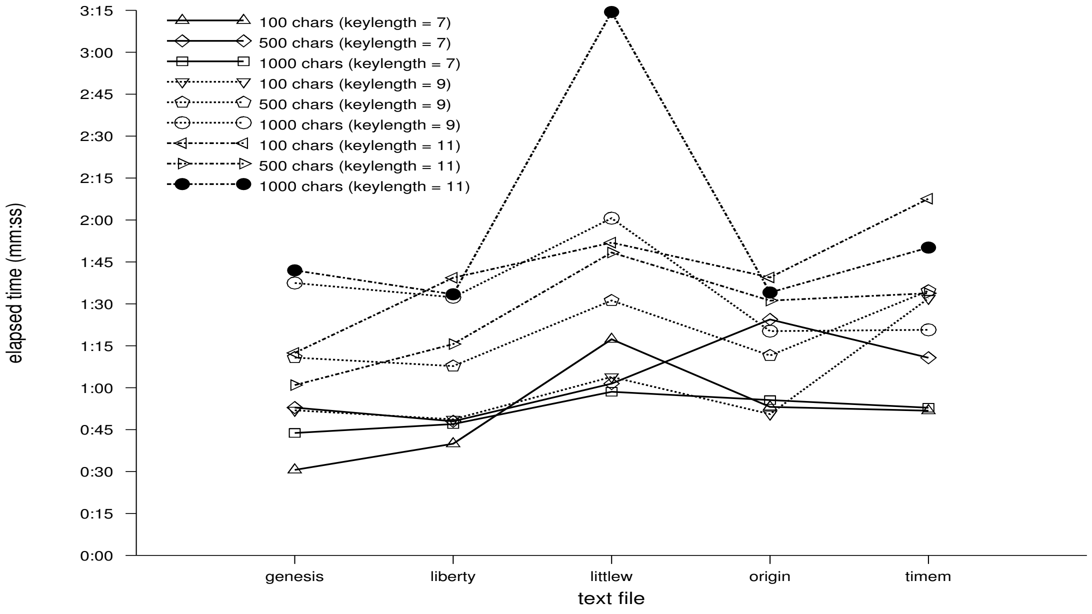

Figure 6.4: Classical Transposition Attack

7; there were no successes at a keylength of 9 or 11. The per-text percentage of complete breaks at key length 7 ranged from 0 % to 20 %, with most texts in the 2 to 4 % range. Mutation and mating rates had little effect on decryption success, with more decryptions occurring at larger population sizes and/or more generations.

The Gründlingh & Van Vuuren attack was also only successful at a keylength of 7. Per-text decryption percentages at key length 7 ranged from 0 % to 28 %. The 100 character text segments failed almost uniformly - there was one text of the five that was broken once, giving it a success rate of 0.416 % as opposed to 0 %. Also, Chapter 1 of Genesis failed to decode at any text length. The other 500 and 1000 character text segments achieved a success rate of 20 % to 28 %. Mutation rate and type had no noticeable effect, and more decryptions occurred with larger population sizes and/or more generations. A population size of 50 resulted in decryption most of the time, as compared to the other population sizes (10, 20, 40).

In comparison to the other attacks, Clark's attack of 1994 [4] was wildly successful. A block length of 2 always decrypted, while block lengths of 4 and 6 had success rates ranging from 5 % to 91 %. Most of the per-text decryption percentages at block length 4 were in the high seventies or low eighties, while the percentages at block length 6 were usually in the mid-forties. Overall, 66 % of the 100 character text tests, 73.5 % of the 500 character text tests, and 74.75 % of the 1000 character text tests were correctly decrypted. As above, larger population sizes and/or more generations increased the decryption rate, with no obvious effect from mutation rate. Block length 6 texts were especially sensitive to the number of generations and population size, failing most of the time on a population size of 10 or 20.

The effects of varying the population size and number of generations in this attack (Clark 1994 [4]) are shown in the following graphs. The first set of graphs, Figures 6.5 through 6.8, show the effect of the number of generations on a specific population size. This effect is independent of the block length used. Each graph shows all three block lengths (2, 4, & 6), but the times are so close as to form one line when displayed on the same graph. This set of graphs also illustrates the minimal effect of the mutation rate. Each group of tests at 25, 50, 100, 200 or 400 generations, forms a plateau for time, indicating that varying the mutation rate from 5 % to 40 % has no measurable effect.

The second set of graphs, Figures 6.9 through 6.13, show the effect of the population size at a specific number of generations. Again, this effect is independent of the block length used. Each graph shows all three block lengths, but only in Figures 6.9 and 6.10 can the difference between block lengths be seen. This visibility is due to the tiny range used in these graphs - a maximum elapsed time of 0.65 or 1.30 seconds. Again, the minimal effect of mutation rate on the elapsed time is apparent.

Overall, these graphs show the expected effect of increasing the number of generations or the population size - more time is required due to the larger pool of chromosomes that need evaluation.

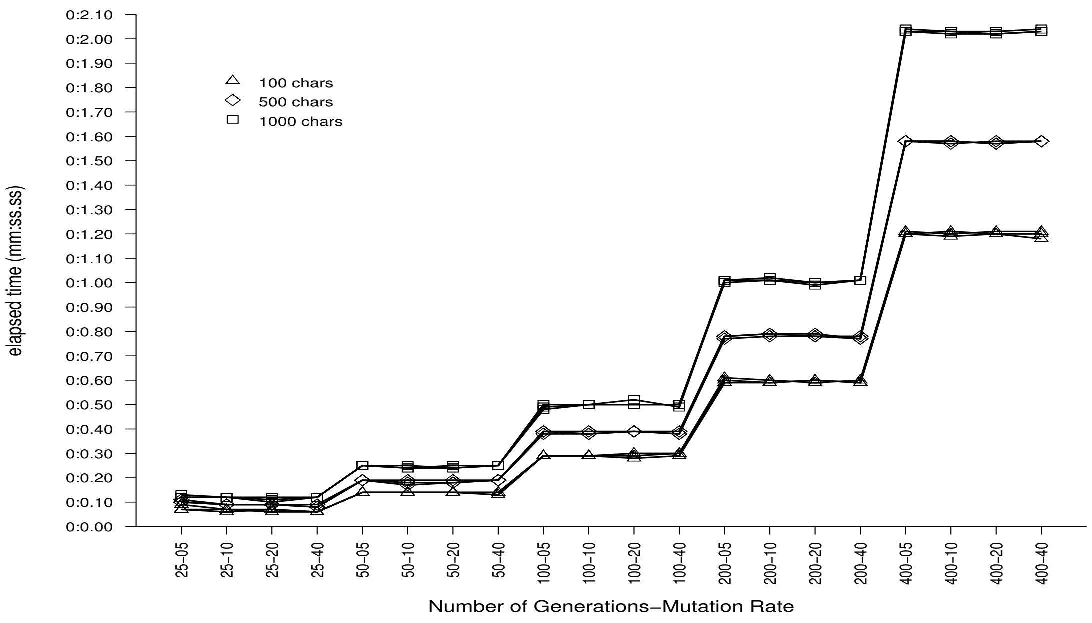

Figure 6.5: Effect of Number of Generations at Population Size 10

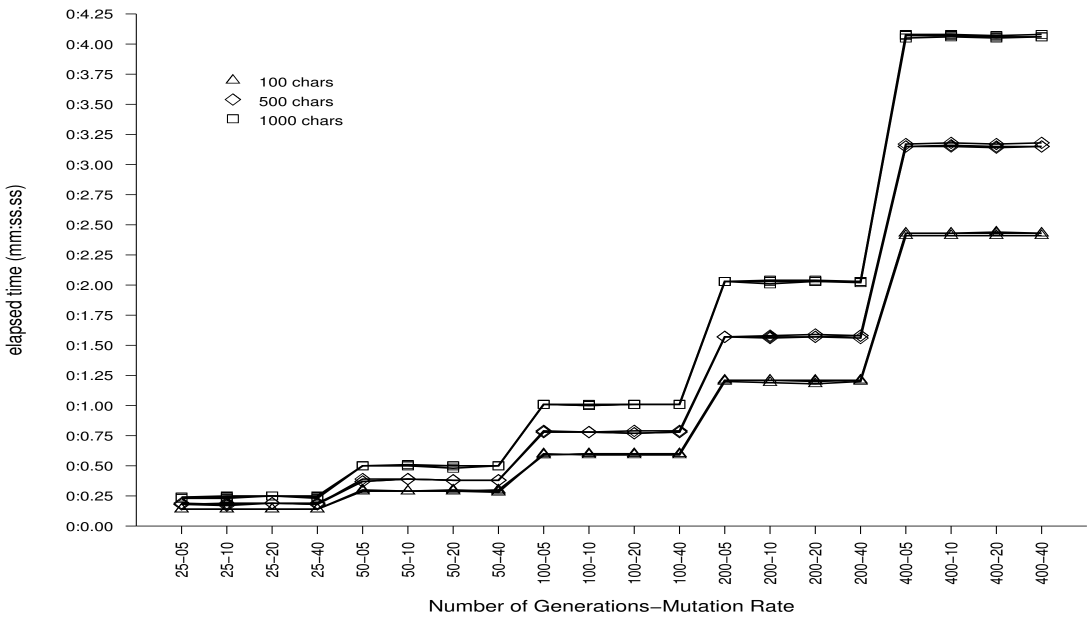

Figure 6.6: Effect of Number of Generations at Population Size 20

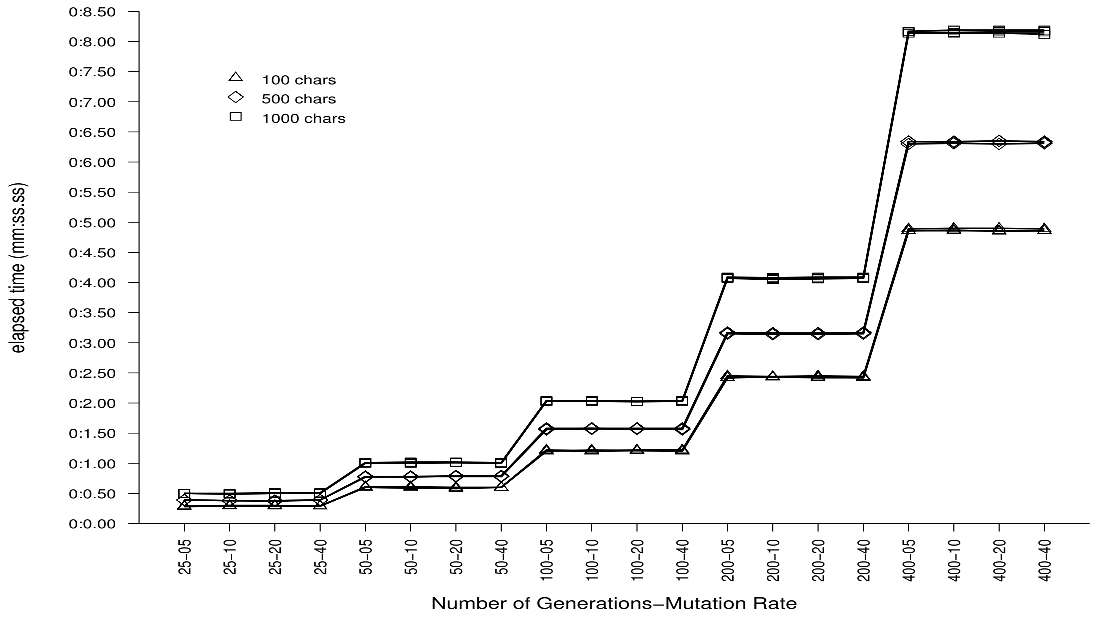

Figure 6.7: Effect of Number of Generations at Population Size 40

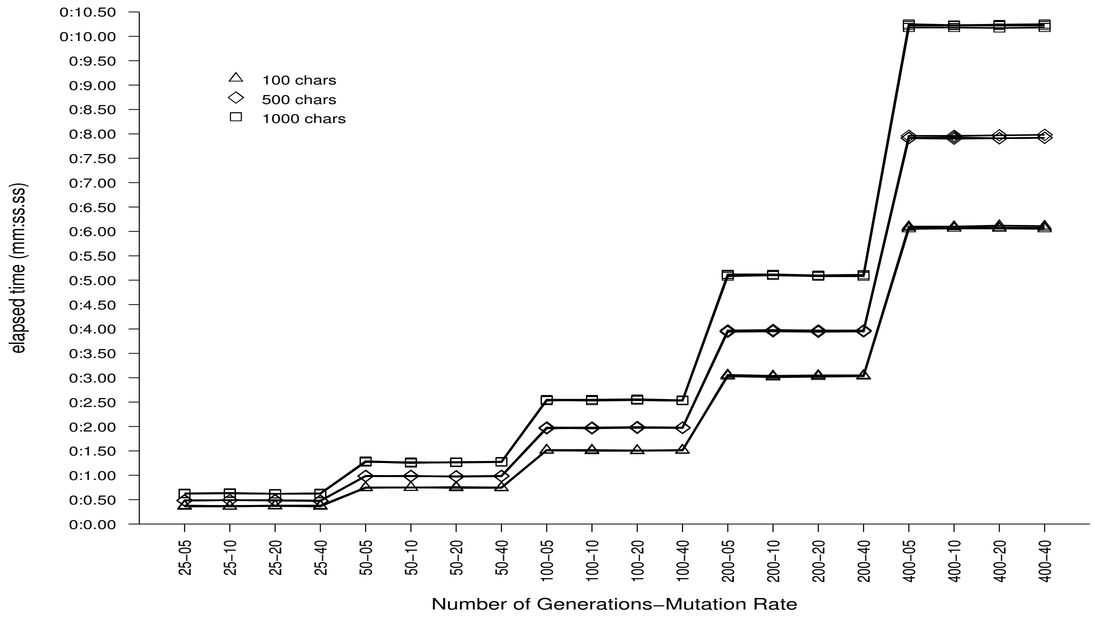

Figure 6.8: Effect of Number of Generations at Population Size 50

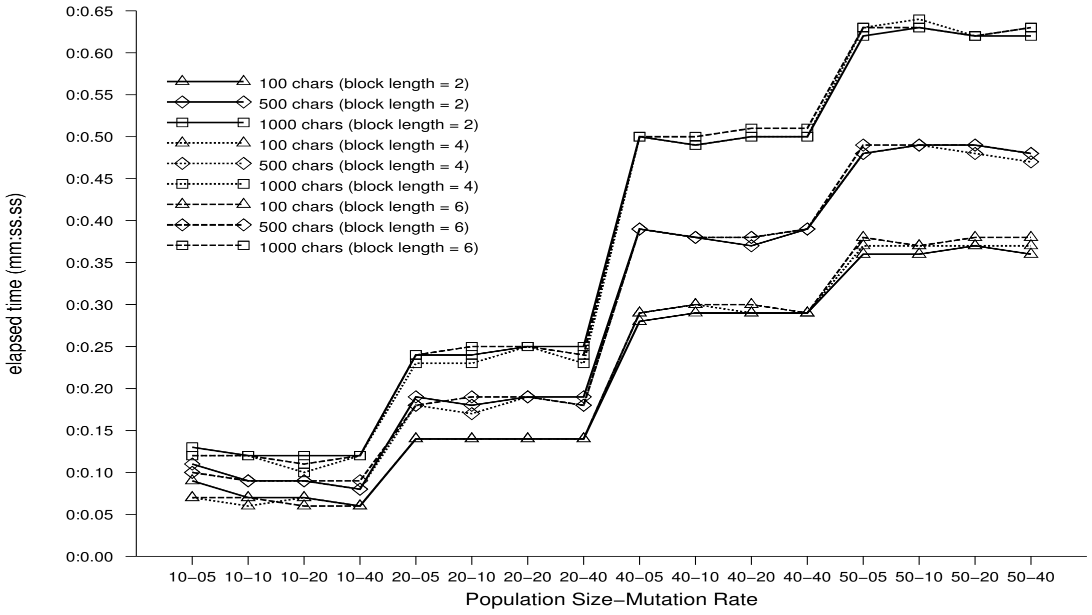

Figure 6.9: Effect of Population Size at 25 Generations

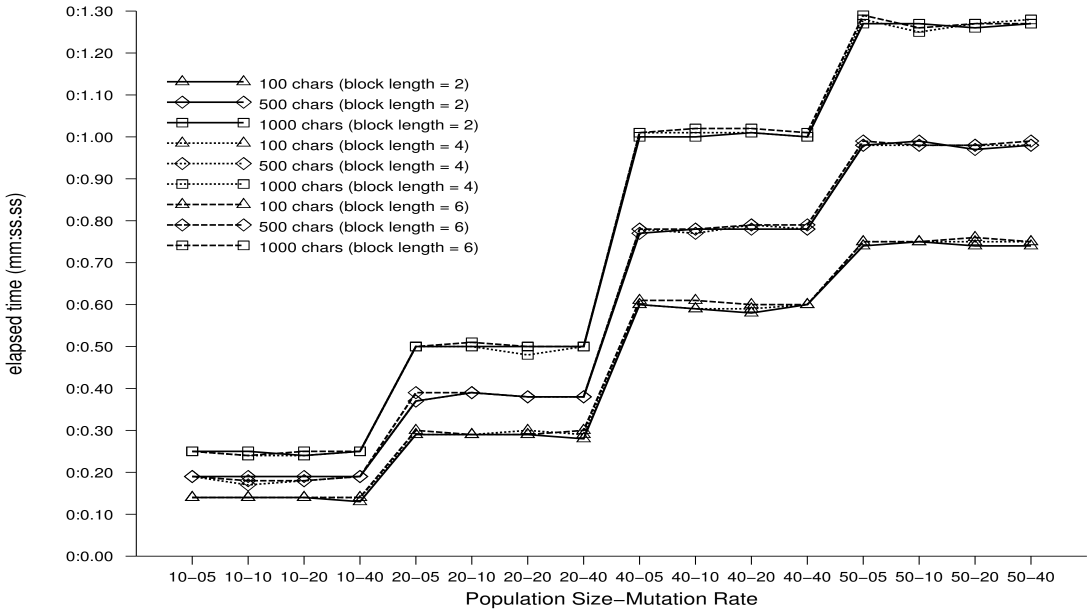

Figure 6.10: Effect of Population Size at 50 Generations

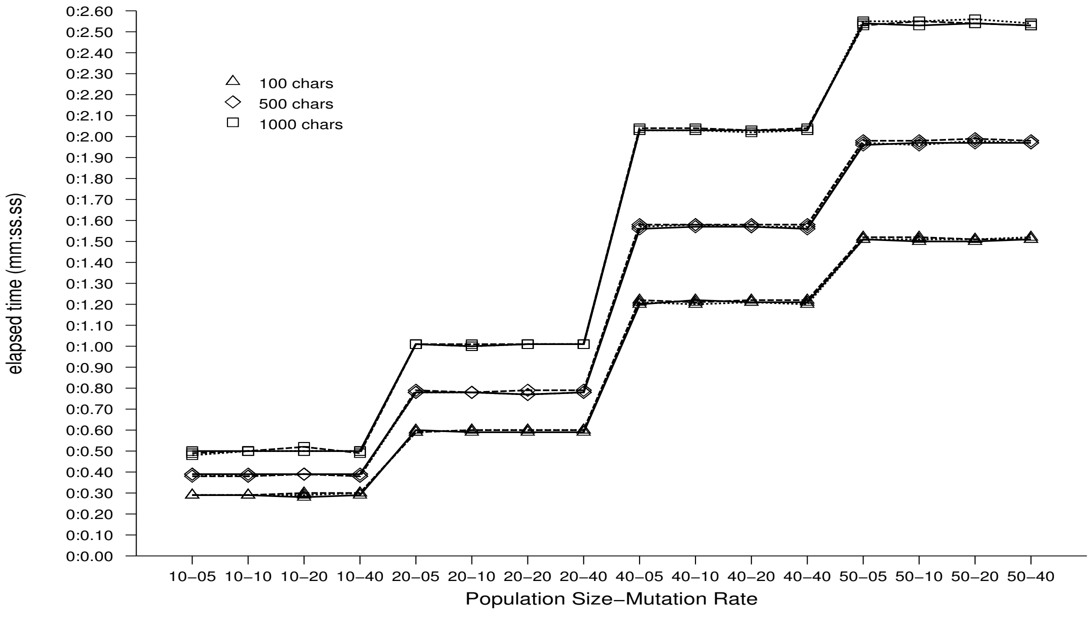

Figure 6.11: Effect of Population Size at 100 Generations

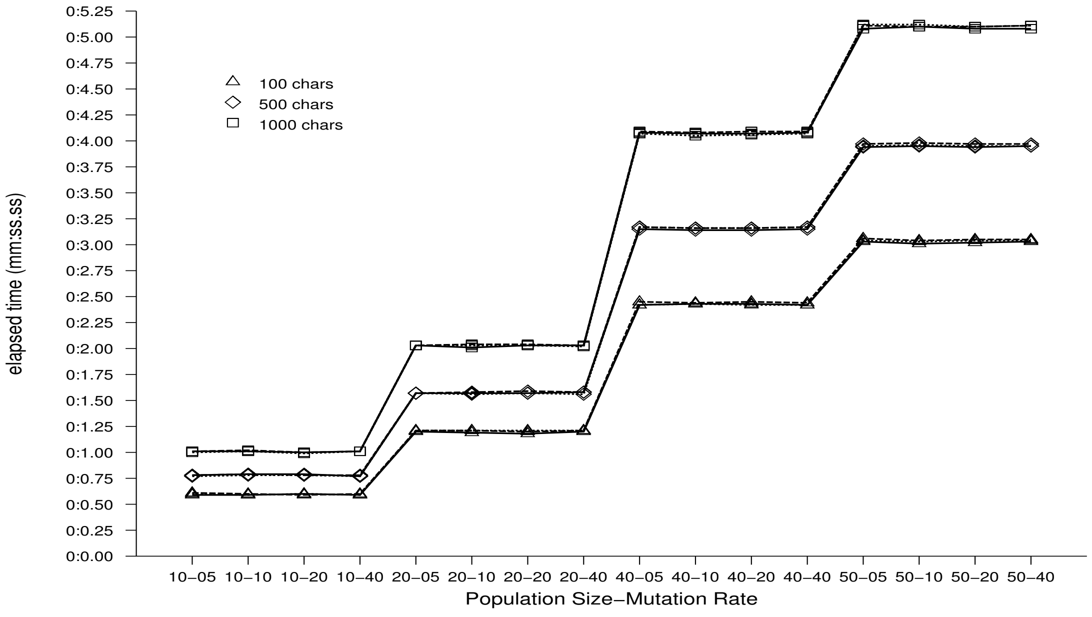

Figure 6.12: Effect of Population Size at 200 Generations

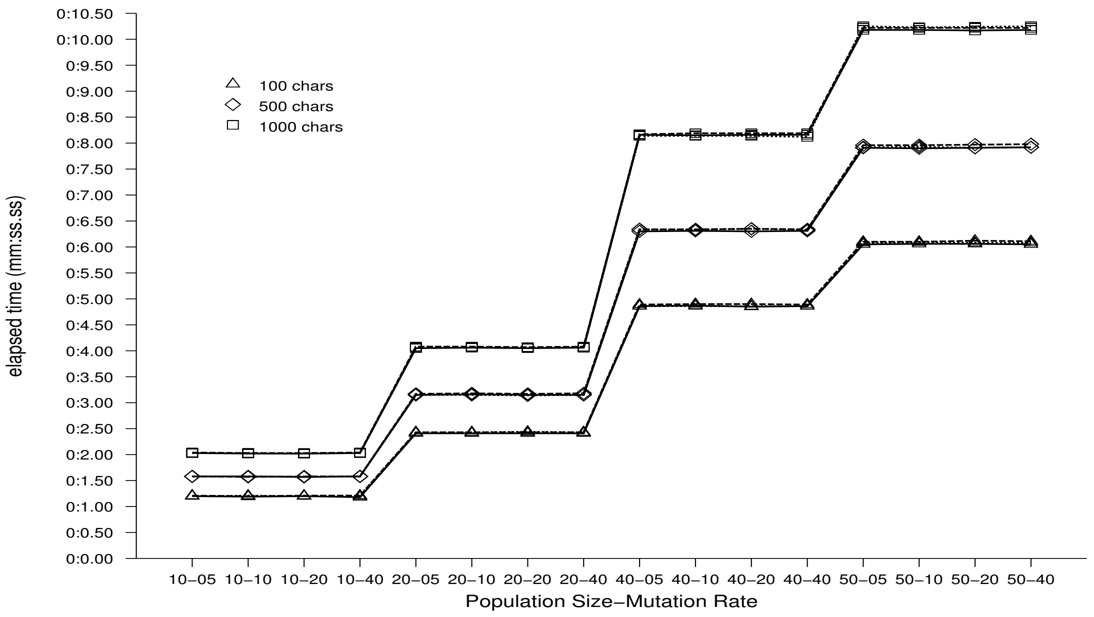

Figure 6.13: Effect of Population Size at 400 Generations

6.4 Comparison of Results

Overall, the GA-based methods were unsuccessful. Traditional methods took more time, but were always successful. Only three of the seven GA-based attacks re-implemented achieved any success. The successful attacks were on the permutation and transposition ciphers.

The traditional attack on the transposition cipher produced elapsed times ranging from :30 to 3:15, with certain texts fairly consistently taking more time. It was successful 100 % of the time, as is standard with a classical attack run to completion.

The two GA-based attacks on the transposition cipher were by Matthews [20] and Gründlingh & Van Vuuren [13]. Both GA attacks were only successful at a key length of 7. The Matthews attack achieved success rates of 2 - 4 %, on average. The Gründlingh & Van Vuuren attack averaged success rates of 6 - 7 %.

The Matthews attack took from :00.05 to :06.27 in the tests run on the 100 character texts, from :00.06 to :07.29 on the 500 character text tests, and from :00.08 to :07.46 on the

1000 character text tests. These elapsed time values are dependent on the population size and number of generations used in a given test. There is no correlation between elapsed time and success rate.

The Gründlingh & Van Vuuren attack took from :00.00 (according to the resolution of the measurement tool) to :01.21 in the tests run on the 100 character texts, from :00.00 to :02.27 on the 500 character text tests, and from :00.00 to :02.99 on the 1000 character text tests. Again, there is no correlation between elapsed time and success rate.

The GA-based attacks took much much smaller amounts of time to complete each test run. At the tiny success rates these two attacks achieved, one success generally takes more time to produce than running a traditional attack. At the 2 - 4 % success rate of the Matthews attack, the test which took the longest time (:07.46) would need to be run over and over for a total time of about 6:22 to produce one success. For the Gründlingh & Van Vuuren attack, with the success rate of 6 - 7 %, the test which took the longest time (:02.99) would need to be run multiple times for a total time of about :50. In both of these estimates, it is assumed that success occurs randomly and spread evenly throughout the tests, which is not the case in actual testing - the successful results tend to cluster together. This being the case, it is not likely that either of the GA-based attacks will produce a result faster than the traditional attack, in the typical case.

The classical permutation attack is a brute force, exhaustive search. Times range from :13 to :15 for texts at block length 2, :21 to :38 for texts at block length 4, and 1:21 to 2:25 for texts at block length 6.

The only GA-based attack on the permutation cipher was the Clark (1994) attack [4]. In comparison to the other attacks, this GA-based attack was wildly successful. A block length of 2 always decrypted, while block lengths of 4 and 6 had success rates ranging from 5 % to 91 %. Most of the per-text decryption percentages at block length 4 were in the high seventies or low eighties, while the percentages at block length 6 were usually in the mid-forties. Overall, 66 % of the 100 character text tests, 73.5 % of the 500 character text tests, and 74.75 % of the 1000 character text tests were correctly decrypted.

This attack took from :00.06 to :06.12 in tests run on the 100 character texts, from :00.08 to :07.96 on the 500 character text tests, and from :00.10 to :10.25 on the 1000 character text tests. These elapsed time values are dependent on the population size and number of generations used in a given test. There is no correlation between elapsed time and success rate.

The Clark (1994) attack is very competitive with the traditional attack. The elapsed time required is much smaller, and the success rates offer a excellent chance that this GA-based approach will be faster than the traditional method, on average.

Chapter 7

Extensions and Additions

There were three genetic algorithm attacks that achieved some success in decryption. These attacks were:

- Matthews' [20] attack on the transposition cipher
- Gründlingh and Van Vuuren's [13] attack on the transposition cipher
- Clark's [4] attack on the permutation cipher

Since the Matthews [20] and Gründlingh & Van Vuuren [13] were only slightly successful, these attacks were re-run with larger population sizes and more generations, to possibly improve success rates. The main problem with these two attacks, besides the low rate of success, is that only a key length of 7 was successfully attacked. The Clark [4] attack was very successful originally, so an attempt was made to extend its success to larger block sizes and to apply this attack to a different cryptosystem.

7.1 Genetic Algorithm Attack Extensions

The Matthews attack was successful only for key length 7 - there were no successes at a keylength of 9 or 11. The per-text percentage of complete breaks at key length 7 ranged from 0 % to 20 %, with most texts in the 2 to 4 % range. Mutation and mating rates had little effect on decryption success, with more decryptions occurring at larger population sizes and/or more generations.

The Gründlingh & Van Vuuren attack was also only successful at a keylength of 7. Per-text decryption percentages at key length 7 ranged from 0 % to 28 %. The overall average for success rate was in the 6 % to 7 % range. Mutation rate and type had no noticeable effect, and more decryptions occurred with larger population sizes and/or more generations.

In comparison to the other attacks, Clark's attack of 1994 [4] was wildly successful. A block length of 2 always decrypted, while block lengths of 4 and 6 had success rates ranging from 5 % to 91 %. Most of the per-text decryption percentages at block length 4 were in the high seventies or low eighties, while the percentages at block length 6 were usually in the mid-forties. Overall, 66 % of the 100 character text tests, 73.5 % of the 500 character text tests, and 74.75 % of the 1000 character text tests were correctly decrypted. As above, larger population sizes and/or more generations increased the decryption rate, with no obvious effect from mutation rate.

7.1.1 Transposition Cipher Attacks

The two attacks on the transposition cipher were run with similar parameters. Population sizes of 100 and 200 were used, in combination with 1,000, 5,000, or 10,000 generations. The other parameters for each attack were:

- Matthews
  - A point mutation rate starting at 0.1 and finishing at 0.5
  - A shuffle mutation rate starting at 0.1 and finishing at 0.8
  - A mating rate starting at 0.8 and finishing at 0.5
- Gründlingh and Van Vuuren
  - A mutation rate of 0.10
  - Mutation using both point and shift mutation

As before, the following set of ciphertexts was used: Chapter 1 of Genesis from the King James Version of the *Bible* [2], Patrick Henry's "Give me liberty or give me death!" speech [3], Section 1/Part 1 of Charles Darwin's *The Origin of Species* [10], Chapter 1 of H. G. Wells's *The Time Machine* [30], and Chapter 1/Part 1 of Louisa May Alcott's *Little Women* [1].

Unfortunately, the larger values for the population size and number of generations did not result in any improvement in the results. The success rate for the Matthews [20] attack remained in the 2 % to 4 % range, while the rate for Gründlingh and Van Vuuren [13] was in the 6 % to 7 % range. Again, only texts at a key length of 7 were decrypted.

7.1.2 Permutation Cipher Attack

The attack on the permutation cipher, Clark's 1994 attack[4], was extended to attempt larger block sizes than before. To aid in this attempt, the parameter values were changed, mostly to larger values. The parameters used were:

- Block lengths of 8, 16, and 32
- Population sizes of 50, 100, and 200
- A number of generations of 200, 400, 1000 or 5000
- A mutation rate of 0.001, 0.01, 0.05 or 0.10

The mutation rate parameter was the only one lowered. The mutation rates used in the original versions of the genetic algorithm attacks are atypically high for genetic algorithm applications. More common values are in the 0.001 and 0.01 range, so these values were used in the extension of this attack.

Again, the following set of ciphertexts was used: Chapter 1 of Genesis from the King James Version of the *Bible* [2], Patrick Henry's "Give me liberty or give me death!" speech [3], Section 1/Part 1 of Charles Darwin's *The Origin of Species* [10], Chapter 1 of H. G.

Wells's *The Time Machine* [30], and Chapter 1/Part 1 of Louisa May Alcott's *Little Women* [1].

The attack was erratically successful on a block length of 8, slightly successful on a block length of 16, and unsuccessful on a block length of 32. The ciphertext attacked seemed to play a larger role in the attack's success. With a block length of 8, two ciphertexts were not decrypted at all, 9 ciphertexts were decrypted more than 50 % of the time, and 4 ciphertexts were decrypted anywhere from 12 % to 40 % of the time. The longer texts (500 and 1000 characters) were attacked more successfully than the shortest text (100 characters). For a block length of 16, several successes were seen against two of the 1000 character ciphertexts. No attacks were successful against a block length of 32.

As before, mutation rate had no apparent effect, and a greater rate of success was achieved among larger populations with many generations.

7.1.3 Comments

The two transposition cipher attacks do not justify writing this style of genetic algorithm attack in their current form. Their success rates are stable, but low, even with extraordinarily large parameter values.

The permutation cipher attack is possibly worthwhile at small block lengths, probably under 10 or so. It does not work at larger block lengths in its current form. This approach would need to be evaluated on a case-by-case basis, in order to determine if this genetic algorithm approach will be cost-effective in comparison with a more traditional, brute force approach.

7.2 Additions

A new attack was implemented on the substitution cipher. Two variants of this attack were tested. The difference between the variants was solely in the scoreboard used.

This attack is a conversion of the Clark [4] attack from the permutation cipher to the simple substitution cipher. The only changes made were those necessary to apply the attack to a different cipher.

The original version of this attack used order-based GAs, since each chromosome represented a permutation. Breeding was done using the position based crossover method for order-based genetic algorithms. In this method, a random bit string of the same length as the chromosomes is used. The random bit string indicates which elements to take from parent #1 by a 1 in the string, and the elements omitted from parent #1 are added in the order they appear in parent #2. The second child is produced by taking elements from parent #2 indicated by a 0 in the string, and filling in from parent #1 as before. Two possible mutation types may then be applied. The first type is a simple two-point mutation where two random chromosome elements are swapped. The second type is a shuffle mutation, where the permutation is shifted forward by a random number of places. Either one or both of these mutation types could occur, depending on the initial parameters. The population for the next generation was produced by merging the parent and child populations and keeping only the (population size) best solutions.

In order to apply this attack to a cipher that was not permutation-based, some changes had to be made. The major change made was the conversion of the chromosomes from numerical genes to alphabetical genes. Another change made was to allow both mutation types to occur all the time - the input parameter previously present was removed. This step was deemed reasonable as mutation type had no apparent effect in prior testing. Breeding and new population creation occurred as before.

The difference between the two variants of the new attack is in the scoreboard used. The first variant uses the same scoreboard as the implementation of Clark's [4] attack on the permutation cipher. The second variant adds several additional items to the scoreboard. The original scoreboard contains the digrams and trigrams TH, HE, IN, ER, AN, ED, THE, ING, AND, and EEE. The variant scoreboard adds RE, ON, HER, and ENT, with the same associated values standard on the original digrams and trigrams. These values are $+1$ for digrams and $+5$ for trigrams. TH is $+2$ while EEE is -5.

The original variant was attacked using the following parameters:

- Population sizes of 10, 20, 40, and 50
- Number of generations of 25, 50, 100, 200, and 400
- Mutation rates of 0.05, 0.10, 0.20, or 0.40

The variant with the extended scoreboard was attacked with these parameters:

- Population sizes of 50, 100, and 200
- Number of generations of 200, 400, and 1000
- Mutation rates of 0.001, 0.01, 0.05, or 0.10

Both variants used the following set of ciphertexts: Chapter 1 of Genesis from the King James Version of the *Bible* [2], Patrick Henry's "Give me liberty or give me death!" speech [3], Section 1/Part 1 of Charles Darwin's *The Origin of Species* [10], Chapter 1 of H. G. Wells's *The Time Machine* [30], and Chapter 1/Part 1 of Louisa May Alcott's *Little Women* [1].

Both attacks were completely unsuccessful. None of the ciphertexts were decrypted in any of the tests.

7.2.1 Comments

This approach appeared promising at the outset. It achieved at least some success on both the transposition and permutation ciphers. Additional modification of the parameters may improve performance, but that seems doubtful given the complete lack of success displayed in these tests. Another option would be to further tune the operators to the new cryptosystem.

If this attack had proved successful, it could have served as a springboard to attacking a modern cipher such as DES or AES with a GA. These ciphers are based on substitution and permutation principles. It may have been possible to attack the components of the system with a GA tuned for that component's cipher.

Chapter 8

Evaluation and Conclusion

8.1 Summary

The primary goals of this work were to produce a performance comparison between traditional cryptanalysis methods and genetic algorithm (GA) based methods, and to determine the validity of typical GA-based methods in the field of cryptanalysis.

The focus was on classical ciphers, including substitution, permutation, transposition, knapsack and Vernam ciphers. The principles used in these ciphers form the foundation for many of the modern cryptosystems. Also, if a GA-based approach is unsuccessful on these simple systems, it is unlikely to be worthwhile to apply a GA-based approach to more complicated systems. A thorough search of the available literature resulted in GA-based attacks on only these ciphers. In many cases, these ciphers were among the simplest possible versions.

Many of the GA-based attacks lacked information required for comparison to the traditional attacks. Dependence on parameters unique to one GA-based attack does not allow for effective comparison among the studied approaches. The most coherent and seemingly valid GA-based attacks were re-implemented so that a consistent, reasonable set of metrics could be collected.

The main metric for both classical and genetic algorithm attacks was elapsed time. In the classical algorithm case, the elapsed time was comprised almost completely of time spent interacting with the user, while in the genetic algorithm case, it was comprised almost entirely of processing time. This difference in composition of elapsed time makes it difficult to compare the attacks on a time basis. Therefore, an additional metric was required.

The secondary metric used was the percentage of successful attacks per test set. Success was defined as complete decryption of the ciphertext. This metric was only calculated for the GA-based attacks, as a classical attack will run until the text is decrypted. A GA-based attack, on the other hand, runs for a specific number of generations, independent of the decryption of the ciphertext. Using this metric, as well as elapsed time, gives one a better feel for the usefulness of each attack. The time measurement is irrelevant if the attack is unsuccessful.

These two metrics were chosen so that traditional and GA-based attacks could be compared. Elapsed time measures the efficiency of the attack while the percentage of successful attacks measures how useful the attack is. Both of these metrics are needed to gain a complete picture of an attack.

Of the twelve genetic algorithm based attacks found in the literature, seven were selected for re-implementation. None of the knapsack or Vernam cipher attacks were re-implemented, for varying reasons. Of these seven, only three attacks were successful in decrypting any of the ciphertexts. The successful attacks were those GA-based attacks on the permutation and transposition ciphers. Clark's [4] GA-based attack on the permutation cipher was by far the most successful. Both of the attacks which used a scoreboard for fitness calculation, Matthews [20] and Clark [4], achieved at least some success. The third successful GA-based attack was by Gründlingh and Van Vuuren [13], the only successful attack to use a fitness equation. These three successful attacks were extended to further examine their success rates. The two transposition attacks were tested using much larger population sizes and numbers of generations, while the permutation attack was attempted on larger block lengths. The two transposition cipher attacks obtained about the same success rate as before, while the permutation cipher attack experienced decreased success. Since the scoreboard approach was successful in both cases it was used, this style of GA-based attack was attempted on the monoalphabetic substitution cipher. The attack was based on Clark's [4] attack, simply altered to apply to a different cipher. Unfortunately, this attempt was unsuccessful. No ciphertexts were completely decrypted with either variant attempted.

8.2 Future Work

There are several areas in which this work could be expanded. One of these is in the investigation of the scoreboard approach to fitness calculation. This method achieved the best results of the two methods used, yet does not seem to be in widespread use. Applying this approach to other problems may prove useful.

A comparison of the scoreboard approach to the fitness equation approach in other applications would be useful as well. This would determine whether the scoreboard approach is limited to specific applications only, or if it has widespread applicability.

Another area for investigation is the use of less common fitness calculation measures. The scoreboard approach, for example, proved useful in this application despite the rarity of its use. Use of other seldom-used approaches may prove additionally valuable.

This work was coded in C and used perl/tcsh scripts. Re-working of the code to make it modular and re-usable for other projects would be a useful extension.

8.3 Conclusion

The primary goals of this work were to produce a performance comparison between traditional cryptanalysis methods and genetic algorithm (GA) based methods, and to determine the validity of typical GA-based methods in the field of cryptanalysis.

The focus was on classical ciphers, including substitution, permutation, transposition, knapsack and Vernam ciphers. The principles used in these ciphers form the foundation for many of the modern cryptosystems. Also, if a GA-based approach is unsuccessful on these simple systems, it is unlikely to be worthwhile to apply a GA-based approach to more complicated systems. A thorough search of the available literature resulted in GA-based attacks on only these ciphers. In many cases, these ciphers were among the simplest possible versions.

Many of the GA-based attacks lacked information required for comparison to the traditional attacks. Dependence on parameters unique to one GA-based attack does not allow for effective comparison among the studied approaches. The most coherent and seemingly valid GA-based attacks were re-implemented so that a consistent, reasonable set of metrics could be collected.

The two metrics used, elapsed time and percentage of successful attacks, were chosen so that traditional and GA-based attacks could be compared. Elapsed time measures the efficiency of the attack while the percentage of successful attacks measures how useful the attack is. Both of these metrics are needed to gain a complete picture of an attack.

Of the genetic algorithm attacks found in the literature, totaling twelve, seven were re-implemented. Of these seven, only three achieved any success. The successful attacks were those on the transposition and permutation ciphers by Matthews [20], Clark [4], and Gründlingh and Van Vuuren [13], respectively. These attacks were further investigated in an attempt to improve or extend their success. Unfortunately, this attempt was unsuccessful, as was the attempt to apply the Clark [4] attack to the monoalphabetic substitution cipher and achieve the same or indeed any level of success.

Overall, the standard fitness equation genetic algorithm approach, and the scoreboard variant thereof, are not worth the extra effort involved. Traditional cryptanalysis methods are more successful, and easier to implement. While a traditional method takes more time, a faster unsuccessful attack is worthless. The failure of the genetic algorithm approach indicates that supplementary research into traditional cryptanalysis methods may be more useful and valuable than additional modification of GA-based approaches.

Bibliography

[1] Alcott, Louisa May. *Little Women Parts I & II*. Great Literature Online 1997-2003. Retrieved August 20, 2003 from http://www.underthesun.cc/Classics/Alcott/littlewomen/.

[2] *The Holy Bible*, King James Version. New York: American Bible Society: 1999; Bartleby.com, 2000. Retrieved August 20, 2003 from http://www.bartleby.com/108/.

[3] Bryan, William Jennings, ed. *The World's Famous Orations*. New York: Funk and Wagnalls, 1906; New York: Bartleby.com, 2003. Retrieved August 20, 2003 from http://www.bartleby.com/268/.

[4] Clark, A. (1994). Modern optimisation algorithms for cryptanalysis. In *Proceedings of the 1994 Second Australian and New Zealand Conference on Intelligent Information Systems, November 29 - December 2,* (pp. 258-262).

[5] Clark, A., & Dawson, Ed. (1997). A Parallel Genetic Algorithm for Cryptanalysis of the Polyalphabetic Substitution Cipher. *Cryptologia, 21* (2), 129-138.

[6] Clark, A., & Dawson, Ed. (1998). Optimisation Heuristics for the Automated Cryptanalysis of Classical Ciphers. *Journal of Combinatorial Mathematics and Combinatorial Computing, 28*, 63-86.

[7] Clark, A., Dawson, Ed, & Bergen, H. (1996). Combinatorial Optimisation and the Knapsack Cipher. *Cryptologia, 20*(1), 85-93.

[8] Clark, A., Dawson, Ed, & Nieuwland, H. (1996). Cryptanalysis of Polyalphabetic Substitution Ciphers Using a Parallel Genetic Algorithm. In *Proceedings of IEEE International Symposium on Information and its Applications, September 17-20*.

[9] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). *Introduction to Algorithms: Second Edition*. Cambridge, Boston: MIT Press, McGraw-Hill.

[10] Darwin, Charles Robert. *The Origin of Species*. Vol. XI. The Harvard Classics. New York: P.F. Collier & Son, 1909-14; Bartleby.com, 2001. www.bartleby.com/11/. [August 20, 2003].

[11] Gaines, H. F. (1956). *Cryptanalysis: a Study of Ciphers and their Solutions*. New York: Dover.

[12] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Boston: Addison-Wesley.

[13] Gründlingh, W. & van Vuuren, J. H. (submitted 2002). Using Genetic Algorithms to Break a Simple Cryptographic Cipher. Retrieved March 31, 2003 from http://dip.sun.ac.za/˜vuuren/abstracts/abstr_genetic.htm

[14] Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor: The University of Michigan Press.

[15] Kahn, D. (1967). *The Codebreakers: The Story of Secret Writing*. New York: Macmillan.

[16] Kreher, D. L. & Stinson, D. R. (1999). *Combinatorial Algorithms: Generation, Enumeration, and Search*. Boca Raton: CRC Press.

[17] Kolodziejczyk, J., Miller, J., & Phillips, P. (1997). The application of genetic algorithm in cryptoanalysis of knapsack cipher. In Krasnoproshin, V.; Soldek, J.; Ablameyko, S.; Shmerko, V. (Eds.), *Proceedings of Fourth International Conference PRIP '97 Pattern Recognition and Information Processing, May 20-22,* (pp. 394-401). Poland: Wydawnictwo Uczelniane Politechniki Szczecinskiej.

[18] Lin, Feng-Tse, & Kao, Cheng-Yan. (1995). A genetic algorithm for ciphertext-only attack in cryptanalysis. In *IEEE International Conference on Systems, Man and Cybernetics, 1995,* (pp. 650-654, vol. 1).

[19] Lubbe, J. C. A. van der. (1998). *Basic Methods of Cryptography*. (S. Gee, Trans.). Cambridge: Cambridge University Press. (Original work published 1997).

[20] Matthews, R.A.J. (1993, April). The use of genetic algorithms in cryptanalysis. *Cryptologia, 17*(4), 187-201.

[21] Menezes, A., van Oorschot, P., & Vanstone, S. (1997). *Handbook of Applied Cryptography*. Boca Raton: CRC Press.

[22] Seberry, J. & Pieprzyk, J. (1989). *Cryptography: An Introduction to Computer Security*. Sydney: Prentice Hall.

[23] Shamir, A. (1982). A polynomial time algorithm for breaking the basic Merkle-Hellman cryptosystem. In *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science, 1982*, (pp. 145-152).

[24] Sinkov, A. (1968). *Elementary Cryptanalysis: A Mathematical Approach*. New York: Random House.

[25] Spillman, R. (1993, October). Cryptanalysis of knapsack ciphers using genetic algorithms. *Cryptologia, 17*(4), 367-377.

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

[27] Stinson, D. (2002). *Cryptography: Theory and Practice*. Boca Raton: CRC Press.

[28] Oliver, I.M., Smith, D.J., & Holland, J.R.C. (1987, July). A Study of Permutation Crossover Operators on the Traveling Salesman Problem. In John J. Grefenstette (Ed.), *Proceedings of the 2nd International Conference on Genetic Algorithms, Cambridge, MA, USA, July 1987*, (pp. 224-230). Lawrence Erlbaum Associates.

[29] Tomassini, M. (1999). Parallel and Distributed Evolutionary Algorithms: A Review. In K. Miettinen, M. Mäkelä, P. Neittaanmäki and J. Periaux (Eds.), *Evolutionary Algorithms in Engineering and Computer Science* (pp. 113 - 133). Chichester: J. Wiley and Sons.

[30] Wells, H. G. *The Time Machine*. Bartleby.com, 2000. Retrieved August 20, 2003 from http://www.bartleby.com/1000/

[31] Yaseen, I.F.T., & Sahasrabuddhe, H.V. (1999). A genetic algorithm for the cryptanalysis of Chor-Rivest knapsack public key cryptosystem (PKC). In *Proceedings of Third International Conference on Computational Intelligence and Multimedia Applications, 1999*, (pp. 81-85).