The University of York Department of Computer Science

**Submitted in part fulfilment for the degree of MEng.**

# A Search-Based Tool for the Automated Cryptanalysis of Classical Ciphers

Michael James Banks

5th May 2008

Supervisor: Professor John A. Clark

Number of words = 25,265, as counted by `detex -l | wc -w`.
This includes the body of the report, and Appendix A, but not Appendix B or C.

## Abstract

The field of classical cryptography encompasses various forms of simple pen-and-paper ciphers that were in widespread use until the early 20th century. Although these ciphers have long been surpassed by modern cryptographic systems, they can still be challenging to break using manual methods alone. Indeed, there exist several well-known classically-encrypted cryptograms which, at present, remain unbroken.

Automated cryptanalysis of classical ciphers has been carried out in existing research, using optimisation techniques in conjunction with appropriate heuristics to evaluate the validity of decryptions. However, this work is largely limited to a few kinds of simple ciphers and the results obtained by some researchers have been criticised by others as being suboptimal.

Building on the approaches used by earlier work, a flexible software tool is constructed to perform automated cryptanalysis on texts encrypted with various kinds of classical ciphers. The tool is expressly designed to support the tailoring of cryptanalysis to particular styles of ciphertext, featuring an extensible framework for defining ciphers and supporting different evaluation heuristics and optimisation algorithms.

The efficacy of the tool is investigated using a selection of sample ciphertexts and unsolved cryptograms. Topics for further research into automated cryptanalysis are proposed.

## Statement of Ethics

There is always a possibility that published cryptanalysis techniques will be used for unethical purposes. However, the cryptanalysis techniques described in this report are applicable only to classical ciphers and are unsuitable for performing attacks on modern cryptographic systems.

In the author's judgement, the work described in this report contravenes no ethical guidelines regarding the applications of cryptanalytic techniques.

# Contents

# 1 Classical Cryptography

*This chapter presents an overview of classical ciphers and cryptanalysis techniques and their relationship to modern cryptographic systems. A selection of unsolved cryptographic puzzles are presented.*

## 1.1 Introduction to Cryptography

*Cryptography* (from the Greek *kryptós gráphein*, or "hidden writing") is the study of methods for preserving the secrecy of information.

Whether a message is transmitted by courier, the postal system, telephone or the Internet, there is always a possibility that it will be intercepted by an unauthorised party. If the contents of the message are confidential, then one should take appropriate measures to ensure its secrecy. This is the role of cryptography, which is – and has been for hundreds of years – an essential tool for achieving security in communications.

Over the past 40 years, the global proliferation of digital communication networks, in conjunction with the exponential increase in available computational resources, has revolutionised the field of cryptography. Today, the practical applications of modern cryptographic tools and techniques are wide-ranging, but the objective of all is to establish and preserve – in some way – the following *information security* properties:

**Confidentiality:** The information contained within a message (or, more generally, an encrypted data object) is unreadable to all but the intended recipient.

**Authenticity:** The truthfulness of properties claimed for a message may be verified and confirmed. These properties may relate, for example, to the authorship of the message, its purpose and the time of its creation.

**Integrity:** If the contents of a message are changed during its transmission from sender to recipient – whether by accidental corruption or by deliberate tampering – the modification can always be detected by the recipient.

**Non-repudiation:** The creator of a message cannot dispute the authenticity of the message, nor can the sender deny having sent the message.

Classical cryptography encompasses all pen-and-paper ciphers developed prior to the advent of more sophisticated cryptosystems in the 20th century. Hence, a distinction may be drawn between classical and modern cryptographic techniques, in that the latter are designed to be performed expressly by computer.

Due to their (relative) simplicity, classical ciphers are today regarded as being fundamentally insecure; they have long been surpassed in security and capability by modern cryptographic systems and, therefore, they are rarely used in practice for any serious application. Remarkably, however, the primitive components of classical ciphers – substitutions and transpositions (Section 1.2) – constitute the "building blocks" of modern block ciphers such as DES and AES.

Cryptography should not be confused with *steganography* ("covered writing"), which is the practice of concealing the *existence* of secret information within a medium: invisible ink, the microdot and digital watermarking are examples of steganographic devices. However, steganography (by itself) cannot be relied upon to ensure secure communication, as it cannot prevent an adversary from discovering the existence of a hidden message and proceeding to extract and then read the information concealed therein.

### 1.1.1 Principles of Cryptography

Classical cryptography is synonymous with *encryption*, the transformation of information (the plaintext) to an unintelligible form (the ciphertext); and *decryption*, the process of retrieving information from the ciphertext. When taken together, the algorithms for encryption and decryption are called *ciphers* and constitute a *cryptographic system* (cryptosystem).

**Definition 1** *(Stinson, 2002)*
*A cryptosystem is a five-tuple* $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$*, where the following conditions are satisfied:*

- $\mathcal{P}$ *is a finite set of possible plaintexts*
- $\mathcal{C}$ *is a finite set of possible ciphertexts*
- $\mathcal{K}$*, the keyspace, is the finite set of all possible keys*
- *For each* $k \in \mathcal{K}$*, there is a encryption rule* $e_k \in \mathcal{E}$ *and a corresponding decryption rule* $d_k \in \mathcal{D}$*. Each* $e_k : \mathcal{P} \to \mathcal{C}$ *and* $d_k : \mathcal{C} \to \mathcal{P}$ *are functions such that* $d_k(e_k(x)) = x$ *for every plaintext element* $x \in \mathcal{P}$*.*

A cryptosystem requires a *key parameter* to specify the encryption (or decryption) rule. The encryption key $e_k$ is closely related to the decryption key $d_k$ and, for *symmetric-key* ciphers (including all classical ciphers), if either key is known, the other may be derived in a straightforward manner[1].

A prerequisite of every strong cryptosystem is that it must not be feasible to derive the plaintext from the ciphertext without the key. This is reflected by Kerckhoffs' principle, which states that *"no inconvenience should occur if [the cryptosystem] falls into the hands of the enemy"* (Bauer, 1997) and, implicitly, the secrecy of the key is paramount to the security of the system[2]. Maintaining the secrecy of a key is difficult in practice, which has led to the development of various systems and protocols for key management and distribution.

### 1.1.2 Introduction to Cryptanalysis

*Cryptanalysis* – in a sense, the antithesis of cryptography – is the art and science of recovering the meaning of encrypted information, without the authorisation of the communicating parties and, by extension, without knowledge of the secret key.

The application of cryptanalytic methods to find and exploit weaknesses in a cryptosystem is known as an *attack*. If an attack on a cipher is successful, the cipher is said to be *broken*[3] and should no longer be regarded as secure.

Only a cryptanalyst is qualified to judge the security of a cryptosystem (Bauer, 1997); if attacked by a determined cryptanalyst, weak encryption is no more secure than no encryption whatsoever. Indeed, overconfidence in the security of a cipher system may result in disaster[4].

[1]This contrasts with *public-key* ciphers, which are designed to ensure that, given only a public key, the process of determining the corresponding private (secret) key is an extremely hard computational problem.

[2]This is equivalent to Shannon's maxim: *"the enemy knows the system being used"* (Bauer, 1997).

[3]A cryptanalyst need not focus on obtaining a total break of a cryptosystem – that is, a means of deriving the secret key from a set of ciphertexts – but may wish to achieve lesser goals, such as determining a causal relationship between plaintexts and ciphertexts. Indeed, cryptanalytic techniques are employed by cryptographers, to find and correct weakness in cryptosystems.

[4]The downfall of Mary Queen of Scots (Section A.4) is a poignant historical example.

## 1.2 Classical Ciphers

### 1.2.1 Substitution Ciphers

A substitution cipher defines a total mapping[5] from the plaintext alphabet to the ciphertext alphabet. This mapping (the key) is often expressed as a permutation of the ciphertext alphabet, assuming the lexicographic ordering of the plaintext alphabet. Hence, for a 26 letter alphabet, such as English, there are 26! ($\simeq 4 \times 10^{26}$) distinct substitution mappings (Menezes et al., 1996).

| **Plain alphabet** | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Cipher alphabet** | F | I | Q | L | C | U | O | Z | Y | S | K | G | A | W | B | E | R | N | T | D | J | H | M | V | X | P |

Figure 1.1: An example substitution mapping over the English alphabet.

#### Simple (monoalphabetic) substitution

Encryption by simple (monoalphabetic) substitution is performed by replacing each symbol (or groups of symbols[6]) in the plaintext with the corresponding ciphertext alphabet symbol, as specified by the key. For example, the key shown in Figure 1.1 transforms the message "`attack at dawn`" to the ciphertext "`FDDFQK FD LFMW`". (Notice that the letters 'A' and 'T' are repeated in the plaintext, correlating with the multiple occurrences of 'F' and 'D' at the same positions in the ciphertext.)

Monoalphabetic substitution does not change the statistical characteristics of a message: the frequency (occurrence count) of each letter in any plaintext will equal the frequency of the equivalent symbol in the corresponding ciphertext. Because the highest frequency symbols in the ciphertext will usually represent the most common letters in natural language, one can cryptanalyse a sufficient length of ciphertext by *frequency analysis* (Subsection 1.3.1) to recover the plaintext.

#### Homophonic substitution

Naïve frequency analysis may be defeated by levelling the frequencies of the symbols in the ciphertext. This may be accomplished by homophonic substitution, in which each letter in the plaintext alphabet is mapped to a set of *homophones* drawn from an enlarged ciphertext alphabet of symbols. The number of homophones assigned to each letter should be proportional to the relative frequency of the letter in the plaintext.

Provided that each homophone is used evenly in encryption, the symbol frequencies of the resultant ciphertext will be uniformly distributed. However, the statistical relations *between* letters are not masked by homophonic substitution, which means that an encrypted message remains vulnerable to frequency analysis on groups of symbols in the ciphertext.

#### Polyalphabetic substitution

A polyalphabetic substitution cipher draws from multiple ciphertext alphabets. Encryption is performed by splitting the plaintext into portions, then encrypting each portion with a distinct substitution key. This technique equates to a one-to-many mapping from plaintext to ciphertext symbols, as opposed to the one-to-one mapping of a simple substitution cipher.

The advantage of polyalphabetic substitution is that repeated letters in a plaintext do not correlate in the ciphertext, making the ciphertext invulnerable to standard frequency analysis.

[5]If the plaintext and ciphertext alphabets contain the same number of symbols, then the mapping is bijective.

[6]Substitution ciphers which operate on groups (blocks) of symbols are said to be *polygraphic*. An example of a polygraphic substitution cipher is the Playfair cipher (Section A.6), which operates on letters in pairs.

The canonical form of polyalphabetic substitution is the Vigenère cipher (Section A.3). This takes a *keyword* as the key parameter, which selects the key functions from a set of regularly shifted ciphertext alphabets as shown in Figure 1.2. The keyword is repeated for the length of the message; each letter is then encrypted using the table row indexed by the corresponding keyword letter. For example, assuming that the keyword is "`hello`", then the message "`attack at dawn`" is encrypted as "`HXELQR EE OODR`".

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

Figure 1.2: The *tabula recta* (or Vigenère tableau) of the English alphabet.

The Vigenère cipher was long thought unbreakable[7], – earning it the title *"le chiffre indéchiffrable"* – until Friedrich Kasiski published a procedure for its cryptanalysis in 1863.

### 1.2.2 Transposition Ciphers

A transposition cipher rearranges the letters in a plaintext according to some pattern. The mapping from plaintext to ciphertext is determined by the permutation key $k : \mathbb{Z}_d \rightarrow \mathbb{Z}_d$, which is a bijective function over the integers 1 to $d$ inclusive.

Two common forms of transposition ciphers are *block* transpositions[8] and *columnar* transpositions. Various other types of transposition cipher – such as geometric transpositions ("turning grille" ciphers) – can be described in terms of block and columnar transpositions.

Since a transposition cipher changes the positions of letters – not the letters themselves – the ciphertext will inherit the exact letter frequencies of the plaintext. This characteristic can be used to determine whether a ciphertext has been produced by a transposition cipher: if so, the ciphertext will possess letter frequencies which approximate those of natural language.

#### Block transposition

To perform block transposition, the letters of the message are separated into blocks of size $d$, then the letters in each block are rearranged in the order specified by the permutation key.

[7] Gilbert Vernam realised in 1918 that if the keyword is randomly generated and equals the message length *without repetition* (a *running key*) then the ciphertext produced will be indistinguishable from random data, making cryptanalysis impossible (Mollin, 2005). The caveat of this "one-time-pad" method is that the keyword may only be used once without compromising the cipher's security (Menezes et al., 1996), which gives rise to the problem of distribution of key material to correspondents.

[8] Block transposition is also known as *complete-unit* (Gaines, 1956) or *periodic* permutation (Denning, 1982).

Using the permutation $\langle 3,1,4,2 \rangle$, the plaintext "`fluorescence`" is encrypted to "`UFOL SRCE CEEN`". The original message may be recovered by re-applying the transposition process using the inverse permutation[9] $\langle 2,4,1,3 \rangle$.

In cases where the plaintext does not divide exactly into a fixed number of blocks, the text may be padded to the required length by appending *null symbols*, before encryption is performed. This ensures a *regular* transposition.

#### Columnar transposition

To prepare a message for columnar transposition, the letters are written (in rows) into a matrix of $d$ columns. Each matrix column is assigned an index number, which is equal to the value at the corresponding position in the permutation (expressed as a vector). The ciphertext is obtained by reordering the columns and reading down each column (Figure 1.3).

| 6 | 3 | 2 | 4 | 1 | 5 |
|---|---|---|---|---|---|
| N | O | W | I | S | T |
| H | E | W | I | N | T |
| E | R | O | F | O | U |
| R | D | I | S | C | O |
| N | T | E | N | T | |

Figure 1.3: An example transposition matrix. The permutation key is $\langle 6,3,2,4,1,5 \rangle$; the ciphertext reads "`SNOCT WWOIE OERDT IIFSN TTUO NHERN`".

If the number of letters in the plaintext is not an exact multiple of $d$, there are insufficient letters to fill the last row of the matrix, so the column lengths will be unequal. This may be rectified by padding the message with null symbols (as with block transposition). Alternatively, the ciphertext may be read directly from the uneven columns, preserving the message length, but resulting in an *irregular* transposition. In this case, the mapping from plaintext to ciphertext is not wholly determined by the key, but also by properties of the text itself, such as its length.

### 1.2.3 Product Ciphers

There is nothing to prevent the use of multiple ciphers to perform encryption. A *product cipher* is simply the composition of two or more substitution or transposition "functions": that is, the ciphertext produced by one cipher function is taken as the plaintext by the next function.

By increasing the complexity of the relationship between plaintext and ciphertext, a product cipher provides greater security than a single substitution or transposition. A classic example of a product cipher is the double columnar transposition[10], which was used as a field cipher in both World Wars (Bauer, 1997); a modern example is the substitution-permutation network[11].

A product cipher must be constructed with care. Two simple substitution functions placed together will compose to form only a single substitution function, since their keys will coalesce. Likewise, two consecutive block transpositions with *equal* key sizes are equivalent to a single block transposition. Indeed, if the keys are pairwise inverse, then encryption will result in the *identity transformation*: the ciphertext will be identical to the plaintext in all cases.

[9] To obtain the inverse $k^{-1}$ of a permutation $k$, each number $1, 2, \ldots, d$ is exchanged with the index of the position that it occupies in the vector form of $k$.

[10] Applying multiple encryptions is a tedious, time-consuming and error-prone task to execute by hand, which historically limited the adoption of product ciphers. However, the advent of high-speed digital computers has rendered this concern irrelevant.

[11] Originally proposed by Shannon (1949), substitution-permutation networks are the basis of Feistel ciphers, in which multiple iterations of alternating substitutions and block transpositions are applied (Denning, 1982).

### 1.2.4 Code Systems

In the context of cryptography, a code is a type of cryptosystem that is defined by a *codebook* (a lookup table, or "dictionary") of *codegroups*, each of which represents an arbitrary unit of text, such as a word, phrase, or an entire sentence. A message is encoded by replacing each unit of the plaintext with the equivalent codegroup. For example, using the codebook in Figure 1.4, the message "`attack the hill at dawn`" would be encoded as "`RED BLACK YELLOW`".

| | | |
|---|---|---|
| attack = red | the castle = white | at dawn = yellow |
| defend = green | the hill = black | at noon = pink |
| retreat from = blue | the town = grey | at dusk = orange |

Figure 1.4: An example codebook.

One recurring example of a code system throughout history is the *nomenclator*, which combines a codebook with a substitution cipher, which is used to encrypt any phrase that is not included (as a codegroup) within the codebook (Section A.4, A.5).

Technically speaking, codes are distinct from ciphers: whereas ciphers are functions on fixed-size groups of letters, code systems operate at the semantic (or linguistic) level and generally shorten the length of an encoded message.

Communication by code is necessarily restricted to the phrases defined in the codebook[12]: hence, ciphers have the advantage of generality. However, a well-designed code system may be harder to cryptanalyse than any classical cipher[13], because an encoded message contains fewer structural "clues" than the equivalent ciphertext, which precludes the use of techniques such as frequency analysis.

## 1.3 Cryptanalysis Techniques for Classical Ciphers

In the domain of classical cryptography, the cryptanalyst is concerned only with finding the secret key (and, by implication, the plaintext) for a given ciphertext. This section presents a selection of cryptanalytic techniques suitable for attacking various types of classical ciphers.

### 1.3.1 Frequency Analysis

In natural language, some letters are used more often than others. For a typical sample of English text, the letter 'E' is (usually) the most common[14], followed by 'T' and 'A', whereas the letters 'J', 'Q', 'X' and 'Z' occur rarely. Taking a large quantity of text as a *corpus*, the characteristic frequency of each letter in the alphabet can be identified (Table 1.1).

To crack a simple substitution cipher, the relative frequencies of each ciphertext symbol may be mapped onto the set of characteristic frequencies for the plaintext alphabet. However, this process is not foolproof: for a given ciphertext, the most common symbol need not necessarily map to 'E', but it will almost certainly represent a high-frequency letter. When the plaintext

[12]Depending on the domain of usage, this is often adequate. Prior to the Second World War, code systems were used exclusively by every navy in the world, because *"the number of things ships can be ordered to do is somewhat limited, and does not demand a great vocabulary"* (Pratt, 1939).

[13]Cryptanalysis of a code requires the deduction of the meaning of every codegroup in the message and, in doing so, the codebook is reproduced. This is akin to translating a document written in an unknown language; indeed, cryptanalytic techniques have been used to "decipher" ancient scripts written in lost languages (Singh, 2000).

[14]This is not always the case: the frequency characteristics of a text may be purposefully skewed, as wryly demonstrated by Singh (2000): *"From Zanzibar to Zambia to Zaire, ozone zones make zebras run zany zigzags."*. (This form of writing is known as a *lipogram*.)

| **Letter** | A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Frequency (%)** | 8.2 | 1.5 | 2.8 | 4.3 | 12.7 | 2.2 | 2.0 | 6.1 | 7.0 | 0.2 | 0.8 | 4.0 | 2.4 |
| **Letter** | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| **Frequency (%)** | 6.7 | 7.5 | 1.9 | 0.1 | 6.0 | 6.3 | 9.1 | 2.8 | 1.0 | 2.4 | 0.2 | 2.0 | 0.1 |

Table 1.1: Characteristic letter frequencies of English, sampled from newspapers and novels. Reproduced from Beker and Piper (1982).

identities of the common ciphertext symbols have been correctly identified, the remaining symbols can be determined by applying a combination of intuition and phonetic knowledge[15].

To cryptanalyse more sophisticated ciphers, frequency analysis may be extended to groups of consecutive letters. (In the cryptographic lexicon, letter groups are known as *n-grams*: single letters are *unigrams*, pairs of letters are *bigrams* and letter triples are *trigrams*.)

Again, each bigram and trigram has a characteristic frequency: in English, the bigram 'TH' and trigram 'THE' are very common (and would be expected to occur many times in a correct decryption), whereas the occurrence of a trigram such as 'ZQT' suggests an erroneous decryption (Clark, 2003).

The effectiveness of frequency analysis is dependent upon the length of the ciphertext. If the ciphertext is too short, the extracted *n*-gram statistics will not correlate with the frequency statistics of natural language, thus yielding little insight into the mapping between plaintext and ciphertext. Churchhouse (2001) suggests that, to cryptanalyse a simple substitution cipher, a minimum of 200 letters is necessary if unigram statistics are used alone, but if bigram and trigram statistics are also factored into the analysis, this may be reduced to 50 or 60 letters.

### 1.3.2 The Kasiski Examination

The Kasiski examination[16] is a tool for determining the period (the number of substitution alphabets) used by a polyalphabetic substitution cipher. As with frequency analysis, this technique relies on there being a sufficient amount of available ciphertext.

First, the ciphertext is examined to identify repeated groups of letters. It is highly likely[17] that these identical ciphertext sequences correspond to multiple encipherments of the same word. Since polyalphabetic substitution is periodic, the distance between each letter group and its repetition must be some multiple of the keyword length. By isolating several repeated sequences, the actual keyword length may be determined as a common factor of the distances between them.

Once the keyword length is known, the ciphertext can be broken into its constituent substitution alphabets, then techniques such as frequency analysis may be applied to each individual alphabet to crack the whole cipher.

A somewhat similar (but more generally applicable) technique is *coincidence counting*, introduced by William F. Friedman in 1922.

### 1.3.3 Probable Words

A word or phrase that is known (or suspected[18]) to exist in the plaintext is called a *crib*. By guessing the location of a crib word in the plaintext, the crib may be matched up with its

[15]An an example, in English (and other languages) the letter 'Q' is invariably followed by 'U'. Hence, if the symbol representing 'Q' can be identified in the ciphertext, then so can the symbol for 'U'.

[16]Whilst the Kasiski examination is attributed to Friedrich Kasiski, who published it in 1863, it was independently discovered by Charles Babbage some ten years earlier (Singh, 2000).

[17]This is not always the case: the repetition may be purely accidental or, alternatively, may be due to repeated letters in the cipher keyword (Gaines, 1956).

[18]Often, a set of crib words can be garnered from the context of a message.

encrypted form at the corresponding position in the ciphertext, forming a "crib pair".

Given a crib pair, key information can be inferred by tracing the relationship between the crib and its encrypted form. From this, other portions of the plaintext can be decrypted. If these decryptions make sense and do not contradict existing knowledge of the plaintext content, it is likely that the position guessed for the crib is correct.

Whilst the cribbing procedure can be carried out manually, it is especially well-suited to automation: a crib may be systematically trialled at each possible position in the plaintext, until a valid position is determined[19].

The cryptanalyst's task is made easier if the ciphertext contains punctuation and spacing between words, since these indicate the length and structure of words and sentences. For this reason, non-alphabetic characters are usually removed from the plaintext prior to encryption.

## 1.4 Unsolved Cryptographic Puzzles

Most classical ciphers are trivially weak by modern standards; they are vulnerable to attacks using well-known cryptanalysis methods (Section 1.3) or with the assistance of computers.

Remarkably, a select few cryptograms[20] – some dating back hundreds of years – remain unsolved, despite being subjected to attacks by some of the world's foremost cryptographers and many keen amateur code-breakers alike. Consequently, they are regarded as the most enduring puzzles in cryptography; some have even achieved fame (and notoriety) that extends beyond the cryptographic literature and into the realm of popular culture.

### 1.4.1 The Dorabella Cipher

Sir Edward Elgar (1857 – 1934), the celebrated English composer, was a keen cryptologist – a man equally fascinated with ciphers as with music (Sams, 1970). Combining his interests, he incorporated coded messages into many of his compositions (Jones, 2004).

Elgar's first major orchestral work, the *Variations on an Original Theme* of 1899 (commonly known as the "Enigma Variations") established him as Britain's foremost composer of the day. This work comprises 14 variations in total, each of which depicts a figure in Elgar's life: these are his wife, 12 of his friends and, finally, himself (Kruh, 1998). The Tenth Variation is a tribute to Dora Penny ("Dorabella"), a young woman whom Elgar had befriended, having discovered a mutual appreciation of cycling, football and kite flying.

In July 1897, Elgar sent an enciphered letter to Dora, which has come to be known as the *Dorabella cipher* (Figure 1.5). Dora herself was quite unable to make sense of it and, after Elgar's death, she published it in her memoirs, stating:

> *I have never had the slightest idea what message it conveys; he never explained it and all attempts to solve it have failed. Should any reader of this book succeed in arriving at a solution it would interest me very much to hear of it.*

Given that Elgar was obsessed with cryptograms (Sams, 1997), the cipher is regarded as unlikely to be random gibberish. Instead, it is to be expected that Elgar had intended that Dora would be able to solve it, so the method of encipherment must not be too complex[21].

The message is 87 characters long; the symbols are drawn from an alphabet of single, double and triple arc glyphs, each rotated in eight orientations (Sams, 1970). Analysis of the frequency

[19]The cryptologic "Bombe" machines, designed by Alan Turing in the Second World War to break the German "Enigma" cipher, made extensive use of this technique (Hodges, 1992).

[20]More unsolved cryptograms are described at http://elonka.com/UnsolvedCodes.html

[21]On the other hand, Belfield (2006) speculates that it is a love letter and, perhaps, Elgar deliberately constructed the cipher so that neither Dora, nor anyone else, would be able to break it.

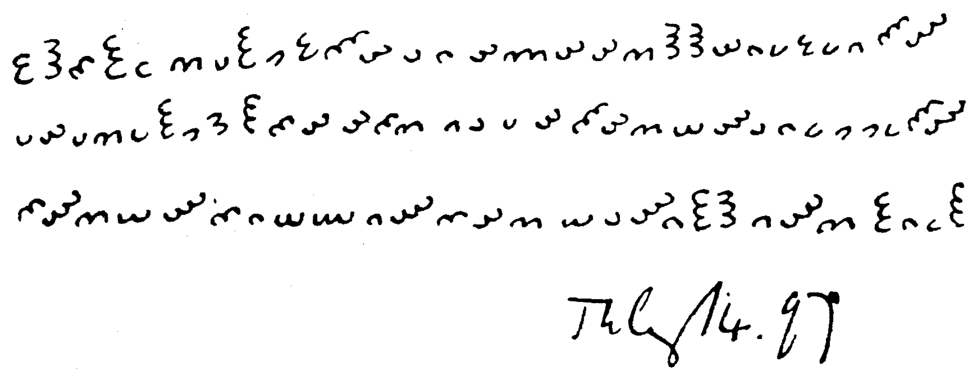


Figure 1.5: Edward Elgar's unsolved cipher letter to Dora Penny.

distribution[22] suggests a substitution cipher: however, all attempts to decrypt it under this assumption have failed (Jones, 2004). There may be a double encryption: a substitution step followed by a transposition step, for instance. Alternatively, some of the characters may represent nulls but, in order to justify which symbols should be disregarded, one must first detect a regular pattern in the message.

Elgar was fond of phonetic spelling[23] and it is possible that he used this technique in the cipher to complicate its decryption. Along these lines, Sams (1970) proposes a solution, which reads as follows:

```
STARTS: LARKS! IT'S CHAOTIC, BUT A CLOAK OBSCURES MY NEW LETTERS.
BELOW: I OWN THE DARK MAKES E.E. SIGH WHEN YOU ARE TOO LONG GONE.
```

This solution appears to be somewhat plausible, but should be treated with caution, as other (equally valid) decryptions could potentially be extracted simply by interpreting the phonetic groups in a different manner.

The Dorabella cipher still receives considerable interest[24], but has not yet given up its secret. To mark the 150th anniversary of Elgar's birth, the Elgar Society has issued a challenge[25] to "Solve the Dorabella Cipher".

### 1.4.2 The Voynich Manuscript

In 1912, the rare book dealer Dr. Wilfrid Voynich bought a chest of ancient manuscripts from the Villa Mondragone, a Jesuit college in Frascati, Italy. Amongst their number, he discovered a small untitled manuscript comprising 102 leaves of vellum. This volume, which has come to be known as the *Voynich manuscript*, is the largest and most famous unsolved text in the public domain, representing the largest prize in code-breaking today (Belfield, 2006).

The manuscript is handwritten in an elegant yet unique script: 99.86% of the text may be represented by 27 basic characters (Landini, 2001). However, the text is unreadable; it is not written in any known language and appears to be enciphered. The only clue to the manuscript's purpose is provided by the esoteric illustrations which cover almost every page, many of which have no apparent parallel with reality. These may be classified into six distinct groups: herbal, astronomical, biological, cosmological, pharmaceutical and "recipes". Analysis of these drawings suggests that the manuscript is of European origin.

[22] This is practically the minimum amount of letters for which frequency analysis is capable of extracting useful information (Subsection 1.3.1).

[23] Elgar referred to Dora's choral singing as "warbling wigorously in Worcester wunce a week" (Jones, 2004).

[24] Discussion continues in the Elgar-Cipher newsgroup, http://tech.groups.yahoo.com/group/Elgar-Cipher/

[25] Elgar Society challenge: http://www.elgar.org/5cipher.htm

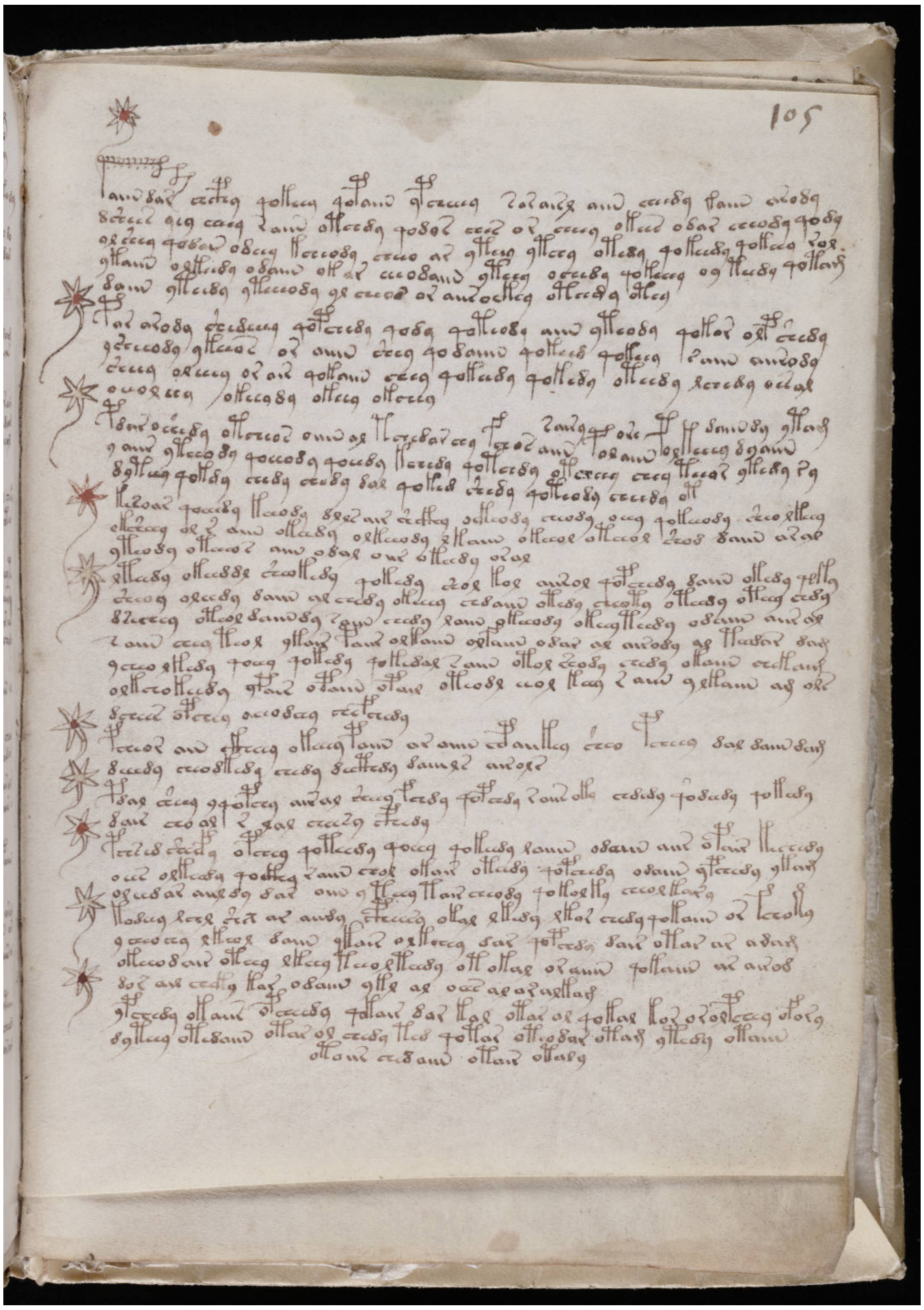


Figure 1.6: Folio 105r of the Voynich manuscript. (Source: Yale University Beinecke Library.)

Attached to the front of the manuscript is a letter[26] written by Johannes Marcus Marci to the Jesuit scholar Athanasius Kircher in 1666. Marci mentions that the manuscript was purchased in 1586 by the Holy Roman Emperor Rudolf II, who believed that it was written by Roger Bacon in the 13th century. However, this has been refuted by modern scholarship, which places the origin of the manuscript between 1450 and 1520 according to the illustrations[27]. The authorship of the manuscript remains a matter of debate.

[26]A transcription of Marci's letter is available at http://www.voynich.nu/letters.html#mm66

[27]Radiocarbon dating of the manuscript has not been carried out, because it would require the destruction of part of the manuscript and would not be sufficiently precise to be conclusive.

### Possible interpretations

The Voynich manuscript has attracted the attention of many professional cryptographers and keen amateurs. A multitude of theories pertaining to its content have been postulated; however, all are based on conjecture, most are unsubstantiated by known evidence and none are conclusive. These theories may be divided into two categories:

**The manuscript is authentic.** It may describe inventions, scientific discoveries, or religious rituals. Alternatively, it could be the handbook of a medieval alchemist or herbalist. In this case, the text must be written in a cipher, an artificial language or an obscure dialect of a natural language. There may even be information concealed by steganographic methods – encoded in the length or shape of pen strokes, for example.

**The manuscript is a hoax.** It may have been fabricated deliberately for financial gain[28], or be the accidental product of the imagination of a disturbed mind[29]. In either case, parts of the manuscript may contain intelligible material, but these must represent the "needle in a haystack" of nonsensical placeholder text.

The earliest "solution" was proposed by Professor William Newbold in 1921. He examined the text of the manuscript under magnification, finding that each character was constructed from a series of minute markings that resembled a form of Greek shorthand. By a complicated process of anagramming and extrapolation, Newbold extracted a Latin plaintext which he claimed to be the work of Roger Bacon. However, close study of the markings showed them to be mere artifacts of the ink drying and, in 1931, John Manly demolished Newbold's solution, by demonstrating that his methods for text extraction were fundamentally flawed and could be used to produced other decipherments with equal validity[30]. Despite their shortcomings, variations of Newbold's methods have been employed by others – such as Joseph Feely in 1943 – but these have never been adequately justified.

The rapid dismissal of many proposed "solutions" has motivated others to favour the hoax hypothesis. Many suspects for fabricating the manuscript have been considered: chief amongst them is Edward Kelley, a notorious Elizabethan-era alchemist and swindler, who may have collaborated with John Dee to manufacture the manuscript as a relic of Roger Bacon in order to defraud Rudolf II (Rugg, 2004).

### Statistical analysis

Textual analysis of the manuscript (Landini, 2001) shows that it is consistent (in form and structure) with natural language: proponents of hoax theories are hence obliged to explain why a forger would consider it necessary to invest much time and effort in generating statistically-sound gibberish. The statistics do not correlate with any specific Indo-European language: Currier (1976) demonstrated the presence of two distinct languages in the text and concluded that the manuscript is the work of multiple scribes, which rules out many of the previous theories for authorship.

If the Voynich manuscript is a genuine document, then there are several possible ways of interpreting the text. The eminent American cryptanalyst William F. Friedman, who undertook

[28]Barlow (1986) claims that the manuscript is a forgery by Wilfrid Voynich himself. However, recently discovered evidence of ownership of the manuscript before 1665 (Zandbergen, 2004) scuppers this theory.

[29]A contemporary example is Dwight Mackintosh (1906 – 1999), "the boy who time forgot", who was institutionalised for most of his adult life. Upon his release in 1978, he conveyed his visions through thousands of surreal drawings, which attracted considerable attention from the art world.

[30]This is reminiscent of the "Bible Code" techniques, which have been applied to religious texts to extract "predictions" of historical and current world events. Quite aside from their statistical fallacies, these techniques can be applied to other texts to produce similar results: see http://cs.anu.edu.au/~bdm/dilugim/moby.html

the earliest computerised studies of the manuscript in the 1940s and 50s, believed that the text was written in an early form of artificial language. A related theory is that the manuscript is encrypted using a simple substitution or transposition cipher[31]. However, this hypothesis is implausible, since the text has withstood the attacks of prominent cryptanalysts for almost a century (Rugg, 2004). An advanced method of encipherment is highly unlikely, because the manuscript text matches the statistical profile of natural language. It is more reasonable to suppose that that the manuscript is encoded; however, without access to the codebook, this cannot be verified.

Recent work by Rugg (2004) demonstrates a method of generating nonsense text with similar properties to natural language by means of a modified Cardan grille cipher. The grille cipher was known in the Elizabethan era, but establishing a connection with the Voynich manuscript is tentative and has been the subject of dispute. Schinner (2007) has applied statistical measures to the Voynich manuscript, showing that the text could have been created by a stochastic process similar to Rugg's method. These results lend credence to the hoax hypothesis and could be seen to substantiate the Kelley theory.

### 1.4.3 The Zodiac Killer Messages

The self-titled "Zodiac" was a serial killer who operated in the Bay Area of San Francisco in the late 1960s. It is known that, between December 1968 and October 1969, the Zodiac murdered five persons and wounded two others; the total number of his victims may be greater[32] (Belfield, 2006). Despite an extensive police investigation, the Zodiac was never caught and his identity remains a mystery.

In a series of letters to the San Francisco press, the Zodiac taunted the authorities for their failure to apprehend him and threatened more murders unless certain demands were met[33]. Four of these messages contain cryptograms, three of which remain unbroken to this day:

**The 408-symbol cryptogram** was sent (in three parts) to the local newspapers in July 1969. Within days, it was solved by amateur cryptanalysts Donald and Bettye Harden, who determined that a homophonic substitution had been used for frequency suppression. The decrypted text is a disturbing insight into the mind of the Zodiac, which details his motivation for murder. The last 18 letters of the deciphered message spell out "`EBEORIETEMETHHPITI`", for which no convincing explanation has yet been produced.

**The 340-symbol cryptogram** was mailed to the *San Francisco Chronicle* in November 1969 (Figure 1.7) and has never been solved conclusively. It features a similar layout and set of glyph forms as the 408-symbol cryptogram, and possesses a related statistical profile[34].

**The "My name is. . . " and "Button" messages** were received in April and June 1970 respectively. These messages contain short cryptograms. The former (13 symbols) proports to contain the Zodiac's identity; in the latter (32 symbols), the Zodiac claims to describe the location of his next planned attack (which was not carried out). Neither cryptogram comprises enough text for effective cryptanalysis and their content remains unknown.

These letters (and the solved 408-symbol cryptogram) are riddled with spelling errors, most of which appear to be intentional. This may be a ploy to frustrate the efforts of code breakers.

[31] If the modern dating of the manuscript is correct, then it must precede the introduction of the polyalphabetic cipher (Section A.3).

[32] The Zodiac is suspected of carrying out several other unsolved murders between 1963 and 1970. He personally claimed a death toll of 37 in his final letters, but some of these may be copycat killings.

[33] Most of the letters start with *"This is the Zodiac speaking. . . "* and all contain his "signature": a circle with a cross drawn through it, resembling the crosshairs of a rifle sight.

[34] A statistical breakdown of the 340-symbol cryptogram is available at http://www.dtm.ciw.edu/chris/z/z2stats.html.

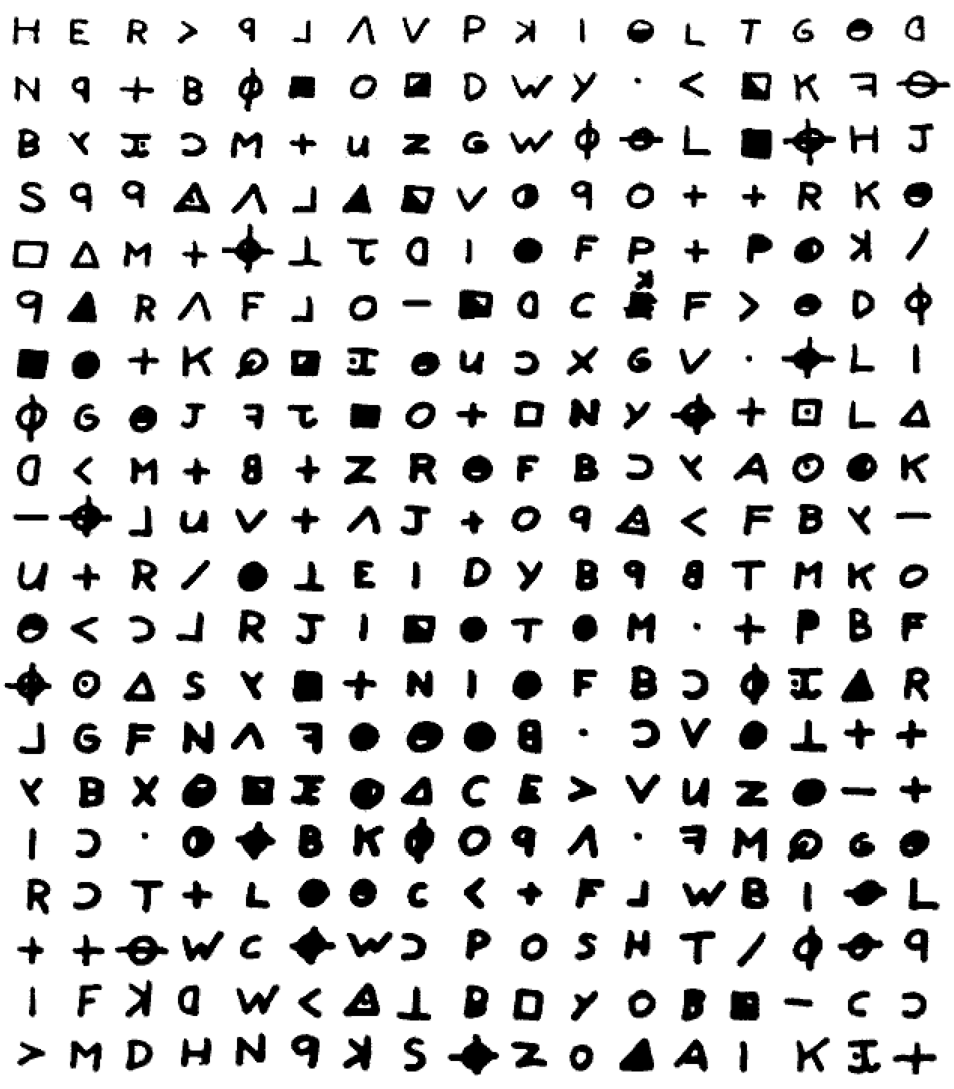


Figure 1.7: The unsolved 340-symbol Zodiac cryptogram.

Clearly, the Zodiac intended his ciphers either to be breakable – as evidenced by the solved cryptogram – or to lead investigators on a wild goose chase. In recent years, the mystery has attracted the attention of many amateur detectives, driving some to the point of obsession.

### 1.4.4 The Potential of a Novel Approach to Cryptanalysis

The cryptograms discussed in Section 1.4 represent a worthwhile target for cryptanalysis; indeed, the extraction of meaningful plaintext from any of these cryptograms would constitute a major achievement. Furthermore, a complete solution would attract considerable interest from outside the cryptographic community.

As stated earlier, many people have attempted to break these cryptograms using conventional cryptanalysis techniques, but none have succeeded in producing a conclusive solution. Assuming that these cryptograms are indeed breakable, this raises the possibility that an original method of cryptanalysis offers a better chance of success for attacking these cryptograms. This topic is revisited in Section 6.5.

# 2 Cryptanalysis as an Optimisation Problem

*This chapter introduces an approach to cryptanalysis that is capable of automatically breaking classical ciphers. Existing work related to this area is reviewed and placed in context.*

## 2.1 Motivation for Automated Cryptanalysis

It was established in Section 1.3 that classical cryptanalysis techniques work by analysing the statistical features of a ciphertext and exploiting them to recover the plaintext. However, these techniques often rely on a human element – ingenuity, or sheer luck – which is not conducive to mechanisation. Furthermore, many cryptanalytic methods are limited to breaking specific classes of cipher and thus cannot be applied if the means of encipherment is unknown.

Research into automated cryptanalysis started in the late 1930s, with the development of the "Bombe" machines to break the "Enigma" encryption used by the German military forces in the Second World War. In 1948, with the embryonic development of the electronic computer in progress, Alan Turing hypothesised that cryptanalysis would be an ideal task for a "thinking machine" (Hodges, 1992). Since then, computing power has revolutionised both cryptography and cryptanalysis.

The simplest and crudest method of automated cryptanalysis is exhaustive search, which systematically checks each key in the (finite) keyspace until a plausible decryption is found. On average, for a keyspace of size $Z$, this "brute force" process will require $Z/2$ evaluations before identifying the correct key. Therefore, it is suitable for breaking trivial ciphers but woefully unsuited for attacking cryptosystems with large keyspaces: assuming a 26-letter alphabet, even a simple substitution cipher has vastly more possible keys than could ever be exhausted by enumeration, given the computing resources available now and for the foreseeable future[1].

However, if the proportion of the keyspace to be searched (for a particular ciphertext) can be drastically reduced by applying an appropriate *heuristic*, then exhaustive search over the remainder of the keyspace may become viable. Hence, in the words of Bauer (1997):

> *The exhaustion attack, although by itself alone rather insignificant, is in combination with other, likewise automatic attacks, the fundamental method of intelligent cryptanalysis.*

For any substitution or transposition function, making a "small" change to the key (such as exchanging two elements) will result in a related (slightly different) encryption. In other words, there is a discernible relationship between the key and the ciphertext or, in the terminology of Shannon (1949), a low level of *confusion*. From this simple observation, it follows that keys that are nearly correct give rise to decryptions that are nearly correct (Clark, 2003) and, hence, *an approximation of the exact key is sufficient to extract readable plaintext*.

## 2.2 Combinatorial Optimisation Problems

The goal of optimisation is to find an *optimal* (best possible) solution for a specified problem, by minimising (or maximising) an *objective function* $f$ over the set $\mathcal{S}$ of all candidate solutions (the *solution space*) to the problem. Combinatorial optimisation is concerned with the domain of optimisation problems in which the set of feasible solutions is discrete.

[1]Modern cryptosystems are explicitly designed to have huge keyspaces, making exhaustive search intractable.

**Definition 2** *(Blum and Roli, 2003)*
*A combinatorial optimisation problem $P = (S, f)$ can be defined by:*

- *a set of variables $X = \{x_1, \ldots, x_n\}$;*
- *variable domains $D_1, \ldots, D_n$;*
- *constraints among variables;*
- *an objective function $f$ to be minimised, where $f : D_1 \times \cdots \times D_n \to \mathbb{R}^+$;*

*The set of all possible feasible assignments is*

$$\mathcal{S} = \{(x_1, v_1), \ldots, (x_n, v_n)\} \mid v_i \in D_i \text{ and } s \text{ satisfies all the constraints}\}$$

The optimal solution set $\mathcal{S}^* \subseteq \mathcal{S}$ may contain multiple (equally optimal) solutions $s^*$:

$$\mathcal{S}^* = \{s^* \mid f(s^*) \leq f(s) \text{ for all } s \in \mathcal{S}\}$$

Most "interesting" instances of the combinatorial optimisation problem – such as the travelling salesman problem – are classed as $\mathcal{NP}$ or $\mathcal{NP}$-hard: in other words, no polynomial-time algorithm is known[2] that finds the exact optimal solution in all instances of the problem. Hence, for sizable problem instances, it is often impractical to use exact algorithms (which guarantee to find the best possible solution) and, instead, one must resort to approximation algorithms to obtain solutions of adequate quality (Clark, 1998).

A *heuristic* is a rule for selecting the "most desirable" solution from a set of alternatives. Heuristic search algorithms, which comprise a class of approximation algorithms, represent a trade-off between the processing time required to obtain a solution and the quality of that solution (with respect to the optimum). The heuristic is employed to reduce the proportion of the solution space to be searched, but ideally retaining an optimal solution.

Heuristics are typically problem-specific and, often, the process of devising an appropriate heuristic for a particular problem is not straightforward. This lack of generality in heuristic search has motivated the development of *metaheuristic*[3] search strategies, starting with simulated annealing (Kirkpatrick et al., 1983).

Metaheuristic search algorithms – discussed in Section 2.4 – are suitable for solving a wide class of optimisation problems; indeed, the challenge is to choose and adapt an existing "off-the-shelf" metaheuristic to a particular problem, rather than developing a specialised metaheuristic from scratch (Gendreau and Potvin, 2005).

Informally, metaheuristics are *"high level strategies for exploring search spaces by using different methods"* (Blum and Roli, 2003). A variety of metaheuristic algorithms have been proposed and, due to their diversity, it is difficult to provide a more concise general definition.

## 2.3 Fitness Functions for Cryptanalysis

A fitness (or cost) function is a scalar-valued measurement of the optimality of a solution to a maximisation (or minimisation) problem[4]. The fitness measure is closely related to the objective function over the solution space: often, a scaling factor is applied to the raw objective

[2]The discovery of an algorithm for solving $\mathcal{NP}$-class problems in polynomial time would be equivalent to a proof of $\mathcal{P} = \mathcal{NP}$. Conversely, if $\mathcal{P} \neq \mathcal{NP}$, no such polynomial-time algorithm exists (Cook, 2000).

[3]The term "metaheuristic", introduced by Glover (1986), combines the Greek *meta* ("beyond") and *heuriskein* ("to find"), implying a more abstract level of search.

[4]Strictly speaking, fitness functions yield higher values for better solutions, whereas cost functions yield lower values for better solutions, but the terms are used interchangeably.

values. If the exact objective function is unknown or impossible to define, a good fitness function will find an approximation of the relative quality of each candidate solution.

In the context of cryptanalysis, a cost function evaluates the suitability ("fitness") of a candidate key for a given ciphertext. The design of most classical ciphers ensures a smooth search landscape: a key that is correct in most of its elements will yield mostly correct decrypts (Section 2.1). The target key – which results in the correct decipherment – should be awarded a minimal cost, corresponding to the global optimum in the search space[5].

A common strategy for evaluating the fitness of a candidate key is to derive a statistical profile from the decrypted text, which is then compared with a "reference" profile extracted from a *corpus* of sample text: the closer the statistics match, the lower the key cost.

This method is only effective when the corpus material possesses a similar profile to the plaintext content, so the corpus must be selected carefully, taking into account the (assumed) language and style of the plaintext. However, Jakobsen (1995) indicates that, in cases where a large amount of ciphertext is available, the corpus statistics need only approximate those of the plaintext.

### 2.3.1 Gram Statistics

The general form of cost functions in the literature, as described by Clark and Dawson (1997), is a weighted linear sum of unigram, bigram and trigram statistics (Equation 2.1). For each $n$-gram[6] $x$ in the plaintext, the observed (decrypted text) frequency $D_x^n$ is subtracted from the expected (corpus material) frequency $K_x^n$. The sum of absolute differences is a measure of the closeness between the observed and expected statistics:

$$Cost(k) = \alpha \sum_{i \in \mathcal{A}} \left| K_{(i)}^{uni} - D_{(i)}^{uni} \right| + \beta \sum_{i,j \in \mathcal{A}} \left| K_{(i,j)}^{bi} - D_{(i,j)}^{bi} \right| + \gamma \sum_{i,j,k \in \mathcal{A}} \left| K_{(i,j,k)}^{tri} - D_{(i,j,k)}^{tri} \right| \tag{2.1}$$

The weightings assigned to unigram, bigram and trigram differences – that is, the values of $\alpha$, $\beta$ and $\gamma$ – are typically determined by experimentation. Changing the relative value of a weight (or other components of the cost function) will affect the evaluation of decrypted texts and, consequentially, alter the search dynamics.

Giddy and Safavi-Naini (1994) report that the introduction of normalisation[7] for each $n$-gram considerably improves recognition of valid decryptions:

$$Cost(k) = \alpha \frac{\sum_{i \in \mathcal{A}} \left| K_{(i)}^{uni} - D_{(i)}^{uni} \right|}{K_{(i)}^{uni}} + \beta \frac{\sum_{i,j \in \mathcal{A}} \left| K_{(i,j)}^{bi} - D_{(i,j)}^{bi} \right|}{K_{(i,j)}^{bi} + \epsilon} + \gamma \frac{\sum_{i,j,k \in \mathcal{A}} \left| K_{(i,j,k)}^{tri} - D_{(i,j,k)}^{tri} \right|}{K_{(i,j,k)}^{tri} + \epsilon} \tag{2.2}$$

It should be noted that some forms of $n$-gram analysis are unsuitable for cryptanalysing particular types of cipher. A transposition operation does not change the letters of a text, so the unigram frequencies of a plaintext are preserved in the corresponding ciphertext. Thus, no useful information can be inferred from analysing the unigrams; instead, one must consider the relationships between letters, which leads to bigram and trigram analyses.

As shown by Figure 2.1, the unigram weighting $\alpha$ should not exceed $\beta$ or $\gamma$, otherwise the cost function's performance will be reduced severely. However, the weight values may be changed as the search progresses: the method outlined by Jakobsen (1995) is – in effect – to identify a key with optimum unigram frequencies and then to incrementally modify this key with respect to bigram statistics.

[5]This is rarely the case in practice, because the target plaintext is unlikely to match exactly the statistical profile that guides the search.

[6]An $n$-gram is a unit of text (a group of consecutive letters) of length $n$ (Subsection 1.3.1).

[7]The $\epsilon$ term is a small value introduced to prevent division by zero when $K_x^n = 0$.

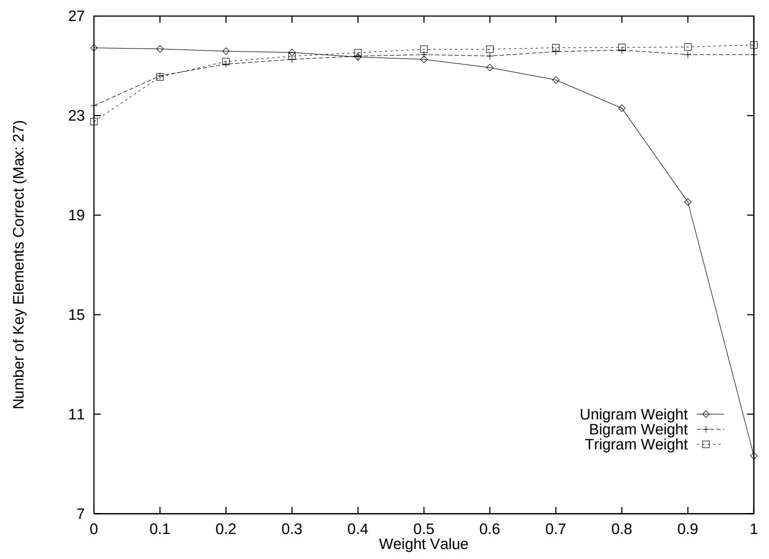


Figure 2.1: Results for cost functions with varying weights, where $\alpha + \beta + \gamma = 1$. Reproduced from Clark (1998).

Clark (1998) compares the performance of three cost functions based on unigram, bigram and trigram frequencies exclusively, to cryptanalyse a simple substitution cipher. The results (Figure 2.2) show that, although trigram statistics are most effective (above a threshold amount of ciphertext), the benefit of using trigrams instead of bigram statistics is minimal.

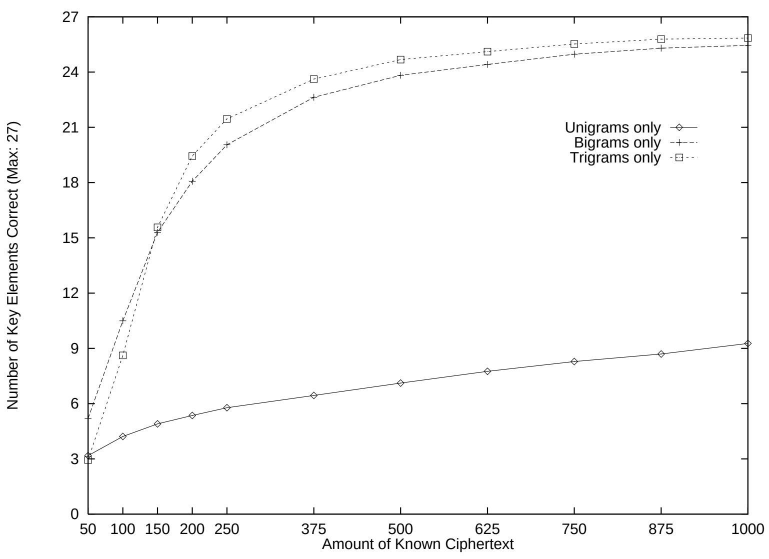


Figure 2.2: Comparative performance of cost functions using only unigrams, bigrams and trigrams, respectively. Reproduced from Clark (1998).

The calculation of trigram statistics is far more computationally expensive than the calculation of bigram statistics[8], so the fitness functions given in the literature are typically limited to analysing unigrams and bigrams for efficiency. However, related keys correspond to decryptions with similar statistics, and Jakobsen (1995) describes how this property can be exploited to reduce the complexity of re-calculating the $n$-gram frequencies by a factor of $n$.

[8]For an alphabet of size $k$, there are $k^n$ possible $n$-grams: ie. for English, 26 unigrams, $26^2 = 676$ bigrams and $26^3 = 17576$ trigrams. (These are upper bounds: the total number of bigrams and trigrams which occur in English text are somewhat lower.) Thus, the number of comparisons required between all observed and expected $n$-grams is exponential in $n$.

### 2.3.2 Counting Specific Bigrams and Trigrams

In contrast to statistical analysis of $n$-grams, Matthews (1993) defines a fitness function that awards "points" for each occurrence of specific bigrams and trigrams (Table 2.1) in a decryption. This method works well for breaking a columnar transposition cipher, because a decryption that scores highly is strong evidence that most of the columns are lined up correctly. To prevent "over-fitting" the statistics, decryptions that contain the trigram 'EEE' (unknown in English) are penalised by the deduction of points[9].

However, this method is not suitable (by itself) for cryptanalysing substitution ciphers because, in the best case, it will correctly identify only those letters that appear in the set of point-scoring bigrams and trigrams, but will not provide any feedback about other letters.

| **Bigram / Trigram** | TH | EE | IN | ER | AN | ED | THE | ING | AND | EEE |
|---|---|---|---|---|---|---|---|---|---|---|
| **Point score** | $+2$ | $+1$ | $+1$ | $+1$ | $+1$ | $+1$ | $+5$ | $+5$ | $+5$ | $-5$ |

Table 2.1: Scoring system for bigram and trigram occurrences, developed by Matthews (1993).

### 2.3.3 Detecting Dictionary Words

The dictionary heuristic – described by Russell et al. (2003) – detects the presence of plaintext sub-strings (words) in a decrypted text and rewards the decryption accordingly. This heuristic is based on the reasoning that, if a complete word is recognised in a decryption, there is a high probability that those elements of the key which correspond to the word are correct. (This idea is loosely related to the "cribbing" procedure outlined in Subsection 1.3.3.)

Formally, the dictionary heuristic is defined by Russell et al. (2003) as:

$$Dict(k) = \frac{1}{L} \sum_{d} d^2 N_d$$

where $L$ is the length of the ciphertext and $N_d$ is the number of dictionary words of length $d$ that are present in the decrypted text that corresponds to key $k$.

Many incorrect keys will – purely by chance – result in spurious decryptions that contain recognisable words[10]. Given a decryption that contains a whole word, the probability of the word occurring in the plaintext increases with the length of the word[11], so the effect of spurious decryptions can be curtailed by considering dictionary words above a certain length.

By boosting the fitness score of decryptions that contain recognisable words, the dictionary heuristic provides the search with an incentive to preserve those elements of the key which are (believed to be) correct. As Russell et al. (2003) point out, a small modification to a partially-correct key may destroy a long and high-scoring word, causing sharp jumps between the fitness scores of related keys and thus disrupting the continuity of the search space. Since metaheuristic techniques are only efficient for traversing a "smooth" search space, the dictionary heuristic should hence be used in combination with other heuristics, in order to "smoothen out" the disruptions in the search space.

[9] The trigram 'EEE' is unknown in English words, yet three consecutive 'E's may appear from the conjunction of two words, such as "`foresee events`".

[10] For example, a ciphertext "`AMJB`" encrypted with the Caesar cipher may be decrypted to either "`cold`" or "`frog`" with equal validity. Without knowledge of the context of the ciphertext, one cannot distinguish between these alternative decryptions.

[11] The minimum ciphertext length required to ensure an unambiguous decryption (in all cases) is known as the *unicity distance*. The unicity distance of a cryptosystem is proportional to its complexity (Bauer, 1997).

## 2.4 Cryptanalysis by Metaheuristic Search

The cryptanalysis of classical ciphers has been formulated as a combinatorial optimisation problem in the literature. A study of various metaheuristic search techniques was conducted by Clark (1998), who concludes that *"optimisation heuristics are ideally suited to implementations of attacks on the classic ciphers"*.

It is evident that traversing the keyspace by moving between related keys will result in a gradual transition between decrypts. However, unlike many problems to which optimisation approaches have been applied, the globally optimal key (with the highest fitness score) need not correspond to the correct decryption, due to the approximating nature of the fitness function. This is corroborated by Giddy and Safavi-Naini (1994), who state that:

> *Most failures to decrypt correctly were the result of the global minimum not being the correct decipherment. [. . . ] The simulated annealing algorithm finds the global minimum, but this minimum does not necessarily represent the correct decipherment.*

It is important to note that optimisation-based techniques do not constitute a "silver bullet" for breaking cryptosystems, let alone for solving hard problems in general. A potent illustration is the application of genetic algorithms to knapsack ciphers, as originally performed by Spillman (1993). This topic has received much attention in the literature (Garg and Shastri, 2007; Kolodziejczyk, 1997; Lebedko and Topchy, 1998), even after Clark et al. (1996) demonstrated that the attacks proposed are suitable only for cryptanalysis of trivially-sized ("toy") knapsacks and that, in practice, it is doubtful whether they are capable of scaling up to realistic knapsack problems (Clark, 2003).

### 2.4.1 Local Search Techniques

A local search algorithm explores the search space $\mathcal{S}$ by progressively moving between candidate solutions (states) in $\mathcal{S}$. The set of states reachable from a current state $s$ is defined by its *neighbourhood*:

**Definition 3** *(Blum and Roli, 2003)*
*A neighbourhood structure is a function $\mathcal{N} : \mathcal{S} \to 2^{\mathcal{S}}$ that assigns to every $s \in \mathcal{S}$ a set of neighbours $\mathcal{N}(s) \subset \mathcal{S}$. $\mathcal{N}(s)$ is called the neighbourhood of $s$.*

#### Hill-climbing search

Hill-climbing search (also known as iterative improvement) is a simple and efficient form of local search. Starting with an initial state $x_0$, the immediate neighbourhood of the current state $x_n$ is sampled. The search "moves" to a successor state $x_{n+1} \in \mathcal{N}(x_n)$ (which must be an improvement on $x_n$) according to a user-defined heuristic. Two commonplace heuristics are:

**Simple (weak) hill-climbing:** select any $y \in \mathcal{N}(x_n)$ given that $y$ is fitter than $x_n$.

**Steepest gradient ascent:** compare all states in $\mathcal{N}(x_n)$ with $x_n$ and then select the fittest.

Both of these heuristics will terminate when a locally optimal state is reached, because a transition to a state of lower fitness is never accepted. To reach the global optimum $x_{opt}$, there must exist a "chain" of improving moves from the initial state to $x_{opt}$. Hence, the performance of hill-climbing is strongly dependent on the definition of the fitness and neighbourhood functions and the initial state (Blum and Roli, 2003).

Jakobsen (1995) uses a hill-climbing approach to solve substitution ciphers. An initial key guess $k$ is constructed from unigram analysis of the ciphertext. Two elements of $k$ are

**Algorithm 1** Pseudocode for simple hill-climbing search.

```
$x \leftarrow$ start state $x_0$
**repeat**
  $y \leftarrow$ an unvisited state in $\mathcal{N}(x)$
  **if** $Cost(y) < Cost(x)$ **then**
    $x \leftarrow y$ {accept improving moves}
  **else**
    reject $y$
  **end if**
**until** all nodes in $\mathcal{N}(x)$ evaluated
**return** $x$ {better than all states in neighbourhood}
```

exchanged[12] to form $k'$, which is evaluated according to bigram fitness (Equation 2.3). If $Cost(k') < Cost(k)$, then $k$ is replaced by $k'$, otherwise $k'$ is discarded. The process is repeated until $Cost(k)$ has not improved for a number of iterations.

$$Cost(k) = \sum_{i,j \in \mathcal{A}} \left| K^{bi}_{(i,j)} - D^{bi}_{(i,j)} \right| \tag{2.3}$$

Since the initial $k$ is biased towards the unigram frequencies of expected language, this method will encounter difficulties should the actual plaintext possess an irregular frequency distribution (Subsection 1.3.1).

Examining the results presented by Jakobsen (1995), the effectiveness of hill-climbing in recovering the correct key does not appear to be competitive with that of simulated annealing (Forsyth and Safavi-Naini, 1993) and genetic algorithms (Spillman et al., 1993). This may be attributed to the nature of hill-climbing search, which terminates once a locally optimal solution is reached, whereas other metaheuristic techniques will continue the search, potentially finding a better (or even the globally optimal) solution.

#### Simulated annealing

Simulated annealing is a probabilistic local search algorithm that mimics the annealing process[13] in metallurgy. Modelled on the statistical thermodynamics of metals (Metropolis et al., 1953), simulated annealing was first presented by Kirkpatrick et al. (1983) to solve the combinatorial optimisation problem.

The underlying principle of simulated annealing is that, in order to "escape" from local optima, it is necessary to move to a solution of poorer quality. The probability of accepting a worsening move decreases as search progresses. (Improving moves are always accepted.) This distinguishes simulated annealing from hill-climbing and, in the general case, increases the likelihood that search will reach the globally optimal solution.

The Metropolis acceptance criterion (Equation 2.4) governs the probability that a worsening move is accepted, in a manner dependent upon a "temperature" parameter $T$. For a minimisation problem, the relative quality of a move from $s$ to $s'$ is measured as $\Delta E = Cost(s') - Cost(s)$ and, as $\Delta E \to \infty$, the less likely it is to be accepted. When $T$ is large, virtually all moves are accepted, resulting in a "random walk" of the search space. As $T \to 0$, almost all worsening

[12] Jakobsen (1995) stores the elements in a vector ordered by frequency. Elements which are close together (those with similar frequencies) are exchanged in preference to those further apart.

[13] Annealing alters the properties of a metal, such as its strength and hardness. The metal is heated to a certain temperature (its *annealing point*) then allowed to cool slowly, so that the particles may rearrange to reach their ground state – a highly structured lattice.

**Algorithm 2** Pseudocode for simulated annealing. (Adapted from Clark (2002).)

```
$x \leftarrow$ start state $x_0$
$T \leftarrow$ initial temperature $T_0$
repeat
  for $i = 1$ to $N$ do
    $y \leftarrow$ an unvisited state in $\mathcal{N}(x)$
    $\Delta E = Cost(y) - Cost(x)$ {minimise $\Delta E$}
    if $\Delta E \leq 0$ then
      $x \leftarrow y$ {always accept improving moves}
    else if $E < U(0,1)$ then
      $x \leftarrow y$ {probabilistically accept worsening moves}
    else
      reject $y$
    end if
  end for
  $T \leftarrow T \times \alpha$ {reduce $T$}
until stopping criterion is met
return best state recorded
```

moves are rejected[14], forcing the search to converge towards an optimal solution.

$$P(s \Rightarrow s') = \begin{cases} exp\left(\frac{-\Delta E}{kT}\right) & \text{if } \Delta E > 0 \\ 1 & \text{otherwise} \end{cases} \tag{2.4}$$

After search at temperature $T_n$ has evaluated $N$ states, $T$ is reduced in accordance with a "cooling schedule" and the search continues at the lower temperature $T_{n+1}$. The cooling schedule has a major impact on the performance of simulated annealing (Blum and Roli, 2003): if $T$ is reduced too quickly, search will fail to convergence to the global optimum. To prevent premature convergence – yet produce a solution in reasonable time – it is commonplace to use a geometric scaling model, such as $T_{n+1} = T_n \times \alpha$, where $0 < \alpha < 1$. Other cooling methods, such as logarithmic scaling, may be used instead.

In the literature, simulated annealing has been used to cryptanalyse simple substitution ciphers (Clark, 1998; Forsyth and Safavi-Naini, 1993) and transposition ciphers (Giddy and Safavi-Naini, 1994). Given a sufficient quantity of ciphertext, the algorithm performs well: Giddy and Safavi-Naini (1994) state that *"a success rate of at least 80% can be obtained if the ratio of the cipher length to the cipher period ($c/n$) is at least 20."*

### 2.4.2 Global Search Techniques

What differentiates global search techniques from local search methods is that a *population* of candidate solutions ("individuals") is maintained, rather than a single solution (Blum and Roli, 2003). This enables the exploration of multiple regions of the solution space at once. As search progresses, the population is incrementally refined over a series of *generations* and eventually converges towards an optimal solution.

#### Genetic algorithms

Genetic algorithms (GAs) were initially developed by John Holland in the 1970s and subsequently popularised by David Goldberg in the 1980s.

[14] If $T = 0$, there is no possibility of accepting a worsening move; the process reduces to hill-climbing (Forsyth and Safavi-Naini, 1993).

The inspiration for GAs is borrowed from nature; specifically, the processes of natural selection acting on a population of organisms. Individuals that possess favourable traits are more likely to survive in their environment and, therefore, to reproduce. The genetic material responsible for these traits will be inherited by their offspring and, consequently, will proliferate throughout the population (Dawkins, 1986).

**Algorithm 3** Pseudocode for a simple genetic algorithm.

```
Pop ← initial population of (randomly-generated) individuals
repeat
  rank all individuals in Pop according to fitness
  parents ← probabilistically select n best-ranking individuals in Pop for reproduction
  for pairs of individuals in parents do
    children ← apply crossover operator to parent pair with probability $p_c$
    apply mutation operator to children with probability $p_m$
    replace lowest-ranking individuals in Pop with children
  end for
until stopping criterion is met
return individual with highest recorded fitness
```

Starting from a population of randomly-generated individuals, a GA repeatedly applies three evolutionary operators – *selection*, *crossover* and *mutation* – to evolve the population:

**Selection:** The selection operator probabilistically chooses individuals from the population for reproduction. Selection carries an element of bias, such that high-fitness individuals have a greater likelihood of being chosen than individuals of lower fitness. Hence, the selection filter reflects the concept of "survival of the fittest".

Dozens of selection methods – each with particular characteristics – have been devised by researchers. Of these, *tournament selection* is regarded as being particularly efficient (from a computational perspective) and, as such, is often favoured for use.

**Crossover:** A crossover operator takes genetic material from two or more individuals, which is recombined to produce new "child" individuals. By rearranging genetic material in this way, crossover provides search with the ability to explore the solution space.

**Mutation:** A gene mutation operator alters an individual's chromosome in some "small" way. Hence, mutation introduces variability into a population (Michalewicz, 1996).

The performance of a GA is particularly sensitive to the choice of parameters; in particular, the probability that a selected individual will be subjected to mutation or crossover:

- If the probability of gene mutation ($p_m$) is too low, then insufficient variability is introduced into the population, causing the population to "stagnate" at a local optimum. Conversely, if $p_m$ is too high, any mutation that improves an individual's fitness will rapidly be cancelled out by subsequent mutations, essentially reducing the evolutionary process to random search. Typically, $p_m$ is set to a value between 0.001 and 0.1.
- If the probability of crossover ($p_c$) is too low, the search will not explore the solution space effectively, resulting in premature convergence of the population. High values of $p_c$, in the order of 0.9, are suitable for most optimisation problems.

The choice of other parameters – including the population size (number of individuals) and the number of generations to be computed – represent a trade-off between the portion of the solution space that is explored (and, hence, the quality of the best solution found) and the amount of processing required to execute the GA. For any optimisation problem, some experimentation with the GA parameters is often necessary to obtain the best results.

Genetic algorithms have been regularly used to attack classical ciphers[15], with modest population sizes and generations. The results quoted by Spillman et al. (1993) indicate that approximately 1,000 individuals are required to find the exact key for a ciphertext (of unspecified length) encrypted by simple substitution. Similarly, when Matthews (1993) applied the GA to a 181-letter text encrypted by columnar transposition, good approximations of the correct key were found within 3,000 candidate key evaluations.

## 2.5 Critical Observations

It is difficult to compare the relative performance of metaheuristic techniques described in the literature, due to the variability of numerous factors:

**Choice of fitness function:** as established in Section 2.3, the characteristics of a fitness function will affect the trajectory of search.

Some fitness functions appear suboptimal: Giddy and Safavi-Naini (1994) criticise the lack of normalisation in the function used by Forsyth and Safavi-Naini (1993). The rationale of other functions is unclear: Spillman et al. (1993) propose a cost function in the spirit of Equation 2.1, but the sum of absolute differences is divided by four *"to reduce sensitivity to large differences"* and then raised to the eighth power *"to amplify small differences"*, without further justification. However, Clark (2002) suggests that experimentation with such "fiddle factors" is often necessary to obtain good results.

**Choice of algorithm parameters:** as discussed in Section 2.4, the choice of parameters for an algorithm has a considerable effect on the algorithm's performance and, hence, the quality of results produced. Different parameter values are used in different papers, making direct comparison of results impossible.

**Selection of sample ciphertexts:** several papers (Jakobsen, 1995; Spillman et al., 1993) only present results for a single sample of ciphertext. The exact ciphertext material (or its source) is rarely specified. In addition, attacks on ciphertexts written in languages other than English are not considered, although experiments with different styles of language are performed by Forsyth and Safavi-Naini (1993).

**Selection of corpus material:** the characteristics of $n$-gram statistics are dependent on the text source from which they are extracted. Forsyth and Safavi-Naini (1993) acquire a set of "expected" $n$-gram frequencies from UNIX manuals; their results show that, for short ciphertexts ($< 3000$ letters), correct decryptions are obtained only if the language style of the plaintext is similar to that of the corpus material.

**Availability of computational resources:** the exponential increase in processing speed and memory capacity has facilitated the development, over the years, of more ambitious metaheuristic search techniques. Results quoted in later papers were produced using faster computers; evidently, computational-dependent metrics (such as an algorithm's run-time) are unsuitable for drawing comparisons between metaheuristics.

**Variability of experimental runs:** the results presented in many papers are averages taken from a number of algorithm runs. With the exception of Matthews (1993), most papers omit additional details, such as the variance between runs. Notably, Matthews (1993) compares

[15]Papers describing the use of GAs for cryptanalysis of classical ciphers include Spillman et al. (1993), Matthews (1993), Bagnall (1996), Clark and Dawson (1997) and Clark (1998), amongst others.

the quality of the decrypts obtained by his genetic algorithm implementation against those achieved by undirected (random) search.

Unfortunately, few of the researchers who present original cryptanalytic techniques evaluate their work against the results achieved in previous publications. This is understandable for the earlier-published papers; however, some of the later publications – such as Jakobsen (1995) – do not show even an awareness of previous work.

Perhaps for these reasons, there is little in the way of comparative studies of the effectiveness of different metaheuristic techniques for cryptanalysis. One study, performed by Clark (1998), evaluates attacks based on simulated annealing, genetic algorithms and tabu search, concluding from experimental results that *"each of the algorithms performed (approximately) as well as the other with regard to the ultimate outcome of the attack"*. This may not be a matter for surprise, due to the relative simplicity of the ciphers attacked (Bagnall, 1996; Clark, 2002).

Clark (1998) provides results which show that, for cryptanalysing a simple substitution cipher, a genetic algorithm requires considerably greater computational resources than either simulated annealing or tabu search. This is to be expected, because implementations of GAs incur overheads in processing and maintaining a population of solutions, whereas local search methods work with only a single solution at a time.

Metaheuristic techniques have also been applied to cryptanalyse "hard" cryptosystems, which fall outside the domain of classical cryptography. Of particular interest is the work of Bagnall et al. (1997), who use a genetic algorithm to successfully cryptanalyse a triple rotor machine (a complex form of polyalphabetic cipher). When the attacks were repeated using either simulated annealing or hill-climbing as the search algorithm, most runs failed to identify the target key (Bagnall et al., 1997). This implies that particular metaheuristics are better suited than others to certain cryptanalysis problems. It remains an open question whether population-based metaheuristics offer greater potential than local search techniques for cryptanalysis of complex ciphers.

### Statement of Intent

The existing research covered in this chapter shows that optimisation-based approaches are eminently suitable for attacks on classical ciphers. However, it is apparent that all previous efforts have been geared towards cryptanalysing specific types of substitution and transposition ciphers, rather than classical ciphers in general.

In order to pursue further research into the application of optimisation techniques for automated cryptanalysis, this project shall focus on constructing a cryptanalysis tool intended to be suitable for attacks on all forms of classical ciphers.

# 3 Project Objectives

*This chapter outlines the goals of the project, which are developed in subsequent chapters. An appropriate strategy for software development is selected and justified.*

## 3.1 Motivation for Project

As discussed in Chapter 2, the utility of metaheuristic techniques for cryptanalysis of classical ciphers has been convincingly demonstrated by previous research.

However, in light of the factors discussed in Section 2.5, it is evident that relevant areas of interest have not been adequately explored by existing work. Other topics that merit detailed investigation include the following:

**Text evaluations:** the majority of published work has relied on $n$-gram frequency analysis (or variations thereof) as the sole means of evaluating the validity of a decryption. By incorporating other heuristics for natural language recognition into the fitness function, the search algorithm will be better equipped to distinguish correct (and partially correct) keys from incorrect ones. This may prove beneficial for attacks on ciphers that are exceptionally difficult to break using frequency analysis alone.

**Attacks on complex ciphers:** the work described in earlier publications is geared to attacks on simple substitution or transposition ciphers. Little attention has been given to more complex kinds of classical ciphers, such as polyalphabetic substitution ciphers (such as rotor machines) and product ciphers. These are somewhat more difficult to break and, therefore, pose an excellent challenge for new cryptanalysis techniques.

**Language considerations:** cryptanalysis of non-English ciphertexts has received virtually no attention in the reviewed literature. Facilities to support languages other than English are a prerequisite for performing effective attacks on ciphertexts written in unknown dialects, such as the Dorabella cipher and the Voynich manuscript (Section 1.4).

To address these topics, the author considers that a new investigation into the potential of automated cryptanalysis of classical ciphers by metaheuristic search is justified.

## 3.2 Project Deliverables

The deliverables proposed for the project are as follows:

1. To identify and implement a *generic* model of the components and operation of various kinds of classical cipher systems.

2. To design and build a software tool for the automated cryptanalysis of ciphertexts encrypted with classical ciphers, using metaheuristic techniques to search the keyspace. (In other words, given a ciphertext and cipher specification, the tool shall attempt to reconstruct the corresponding plaintext and cipher key.)

3. To construct a "front-end" user interface for the aforementioned cryptanalysis tool, incorporating facilities to encrypt and decrypt texts with user-provided ciphers, and to monitor the progress of a decryption when cryptanalysis is taking place.

4. To evaluate the abilities of the completed cryptanalysis tool, by running tests on a representative set of sample ciphertexts. Should the tool succeed in recovering the correct plaintexts, then attacks on unsolved cryptograms (described in Section 1.4) shall be attempted as a further avenue of investigation.

## 3.3 Choice of Software Development Model

A large portion of the project – the construction of the cryptanalysis tool – may be viewed as an exercise in software engineering. In keeping with good engineering practice, an appropriate software development process should be selected and followed.

Since the amount of time available for the completion of the project is limited, emphasis must be placed on achieving the project deliverables, rather than adhering to a rigid and formalised structure of work. This rules out the waterfall and spiral models of software development (along with all "heavyweight" development methodologies) and, instead, necessitates the selection of an "agile" model.

Agile software development methods are characterised by a focus on delivering working software and responding to unexpected problems as they occur (Fowler and Highsmith, 2001). One of the earliest (and most popular) agile methods is *extreme programming* (XP), which may be described as the amalgamation of four basic activities: coding, testing, listening and designing (Beck and Andres, 2004).

The XP style will be adopted for the development phase of the project, with the goal of building a feature-complete cryptanalysis tool within the aforementioned time constraints. However, some of the practices associated with XP – specifically, pair programming and the "customer representative" role – are inappropriate to the nature of this project and will, therefore, be omitted.

## 3.4 Use of Existing Software Libraries

Using pre-built components to construct new software products is accepted wisdom in the field of software engineering. This practice of *software reuse* is advantageous to the development of the tool, for the following reasons (Sommerville, 2001):

**Accelerated development:** less time will be required to integrate existing components within the tool than to design, implement and test the equivalent functionality from scratch.

**Effective use of specialist knowledge:** software built by domain experts is expected to provide sophisticated functionality, which could not otherwise be emulated within the time constraints of the project.

**Increased reliability:** few issues are likely to arise with well-tested "off-the-shelf" components.

For these reasons, the tool shall, where possible, incorporate functionality provided by existing libraries and frameworks, in preference to "reinventing the wheel". However, it cannot be expected that all aspects of functionality can be delivered from pre-built components alone.

# 4 Architecture and Design

*In this chapter, the proposed structure of the tool is described and justified. Particular emphasis is given to those aspects which represent an advancement on existing work.*

## 4.1 Architecture of Tool

A software architecture describes the structure and components of a system at a high level of abstraction, encompassing *"the software elements, the externally visible properties of these elements, and the relationships between them"* (Bass et al., 2003).

### 4.1.1 Choice of Architectural Style

The internal organisation of the tool is modelled on the layered architectural style. The separation of functionality into distinct layers simplifies the task of designing and building the tool, as the details of each layer can be worked out in isolation.

Grouping related components of functionality into layers allows *information hiding*: the implementation details of each layer cannot be accessed by the layers above, which restricts the communication between layers to well-defined interfaces. The advantage of this approach is that modifications localised to one layer will not propagate to the higher layers; indeed, an entire layer may be replaced with another without necessitating any alterations to other layers, provided that the interface between layers remains unchanged.

A disadvantage of layered architectures is the introduction of latency into the system implementation, as modules in different layers cannot interact directly, but must instead communicate across the interfaces between layers (Bass et al., 2003). However, for a small-scale project, the reduction in system performance is minimal and the gains in software simplicity and maintainability are more than compensatory.

### 4.1.2 Overview of Tool Architecture

The architecture of the tool is divided into three broadly-defined layers to maximise conceptual clarity[1]. The role of each layer is as follows:

**Cipher layer:** supports the encryption and decryption of texts; defines a mechanism for the construction of ciphers and compatible keys.

**Cryptanalysis layer:** defines a collection of heuristics to recognise the properties of natural language in a decrypted text; provides facilities for the automated cryptanalysis of ciphertexts using metaheuristic optimisation techniques.

**Control layer:** provides a "front-end" interface to the functionality of the cipher and cryptanalysis layers; coordinates the user's interactions with the tool.

A pictorial view of the tool's architecture is shown in Figure 4.1. The function and purpose of the components in each layer, from bottom to top, is described in the following sections.

[1]Whilst it is possible to further decompose the identified layers to obtain a finer level of granularity, the addition of extra layers obscures the conceptual focus of the architecture.

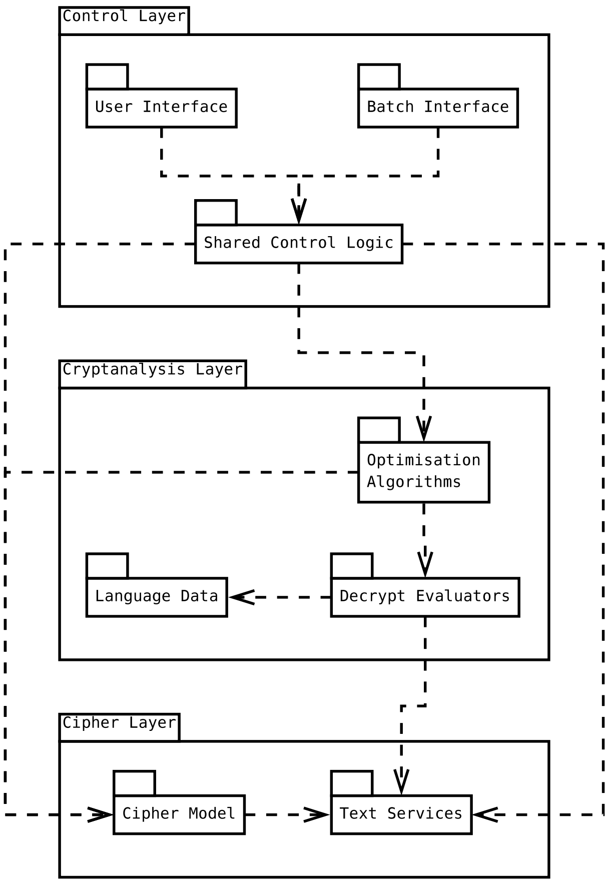


Figure 4.1: The hierarchy of software layers which constitute the tool's architecture. (The dashed lines indicate dependencies between components.)

## 4.2 Cipher Layer

### 4.2.1 Text Services

An *alphabet* is a collection of *letters*. For the purposes of this model, the definition of a letter is relaxed from conventional usage, in that a letter may stand for any symbolic "unit of information" present in a communication medium. This facilitates the construction of alphabets containing any number of letters – such as the (non-standard) letter-forms used in the cryptograms described in Section 1.4 – without resorting to obscure ASCII-like coding schemes.

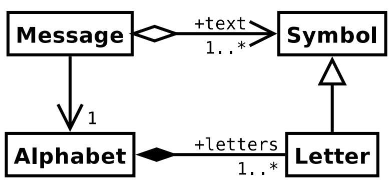


Figure 4.2: The relationship of a message to symbols, letters and an alphabet.

A *message* is composed of letters taken from a single alphabet, together with other non-letter *symbols*, such as spacing and punctuation. (The distinction between letters and non-letters is important to a substitution cipher, which may change only the letters in a message, leaving the non-letters unaltered.) A message may represent either a plaintext or a ciphertext.

### 4.2.2 Cipher Model

#### Functions and Pipelines

A text function is a self-contained unit, which may represent any type of substitution or transposition operation, or any other conceivable form of text transformation. It follows that many (but not all[2]) text functions require a key parameter to specify the exact encryption or decryption rule (Subsection 1.1.1) to be followed.

Text functions may be chained together to build a cipher pipeline (Figure 4.3). A cipher pipeline can encapsulate any number of functions, allowing the construction of all ciphers that can be expressed as a composition of substitution and transposition operations. (In effect, a cipher pipeline models the structure of a product cipher.)

To encrypt a message, each function in the pipeline is applied in turn: the ciphertext produced by one function is piped into the next function as plaintext. Similarly, to decrypt a message, the functions are inverted and applied in reverse order.

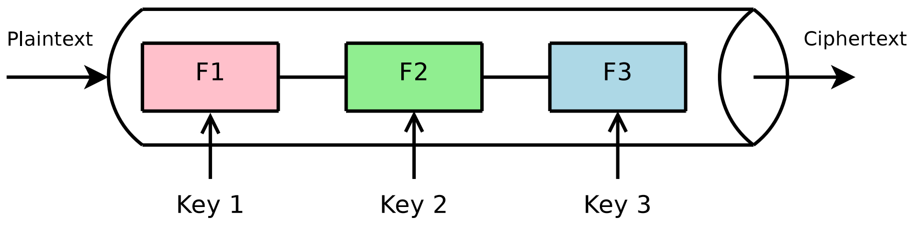


Figure 4.3: An example of a cipher pipeline that encloses three text functions.

The advantage of modelling ciphers as pipelines of functions is that a wide variety of ciphers can be constructed and investigated by using only a small number of generalised substitution and transposition functions. This approach is preferable to building each cipher individually, which would lead to duplication of work.

#### Key Structures

It was shown in Section 1.2 that a key for a classical cipher may be expressed as an indexed sequence of letters or digits. For simple substitutions and transpositions, the sequence must be a permutation over a set of elements: each element may occur once only in the sequence. However, many notable variations of these ciphers – including the Vigenère and Playfair ciphers – expect a keyword (of variable length and without the permutation restriction) to be specified instead. Still other forms of cipher, such as Caesar substitution and rail-fence transposition, require a single integer as the key.

To accommodate these different concepts of a key, each individual definition is implemented as a distinct *key structure*[3]. A key structure encapsulates *key data* (the representation of the key) and conforms to a standard interface, supporting the retrieval and modification of the key data in a uniform manner.

The properties inherent in the definition of a key are enforced by the key structure itself: for example, if the key is required to be a permutation, then no operation on the key data that violates the permutation property will be permitted. Furthermore, modelling keys separately from ciphers enables custom search operators can be defined for individual key structures, as discussed in Subsection 4.3.2.

[2]An example of an un-keyed text function, which has exactly one encryption rule, is the "reversal" operation.
[3]For implementation reasons to be discussed later, a universal "one size fits all" key structure is not suitable.

## 4.3 Cryptanalysis Layer

As discussed in Section 2.4, optimisation by metaheuristic search is a powerful technique for the automated cryptanalysis of classical ciphers. However, the full potential of this approach can only be realised by careful design of the fitness function and the search operators.

### 4.3.1 Evaluation of Decrypts

The fitness function combines two measures of the optimality of a decrypted text:

- at the letter level, the normalised $n$-gram statistical model (Subsection 2.3.1) is used to *minimise* the difference between the decryption and a reference corpus, in terms of unigram, bigram and trigram frequencies. This heuristic is chosen in preference to the "point-scoring" scheme proposed by Matthews (1993), which is only suitable for transposition ciphers (as explained in Subsection 2.3.2).
- at the word level, the dictionary heuristic (Subsection 2.3.3) is used to *maximise* the number of recognisable words that occur in the decryption. Besides the work performed by Russell et al. (2003), the inclusion of this heuristic into the fitness function has not been attempted in earlier work.

These measures may be combined to form a weighted cost function of the form

$$Cost(k) = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

where the $x_i$ terms are the decryption scores and the $w_i$ terms are the associated weights. The relative importance that should be ascribed to each measure will be determined by experimenting with the values of the weights. (As the score of the dictionary heuristic is maximal for correct decryptions, the weight associated with this heuristic will be negative.)

Initially, only the $n$-gram measures will guide the search process, progressively steering towards candidate keys containing a majority of correct key elements[4]. The dictionary heuristic cannot be expected to provide any assistance here, because the occurrence of recognisable words in decryptions with few correct key elements will be merely coincidental.

Once a key with a significant number of correct elements has been reached, it should be possible for the dictionary heuristic to identify some recognisable words in the corresponding decryption. Thereafter – and with the guidance of both heuristics – it is expected that search will be successful in deriving the remainder of the target key and extracting a complete and correct decryption.

### 4.3.2 Identification and Design of Search Operators

A metaheuristic can only effectively explore a problem space if the search operators reflect the properties inherent in the solution space. In the context of classical ciphers, small changes to a key equate to small changes in the fitness score (Section 2.1). To exploit this property, the search operators should be designed to facilitate transitions between similar keys.

Cipher pipelines that contain two or more text functions (i.e. product ciphers) will necessarily require the specification of multiple subkeys to perform a decryption. From the perspective of the optimisation algorithm, search is performed over a single keyspace, which is formed by composing the "sub-keyspaces" associated with each individual function in the pipeline.

[4]The probability of successfully identifying a key with a significant number of correct elements is highly dependent on the cipher's complexity. (Figure 2.2 indicates that approximately 150 ciphertext letters are sufficient to correctly identify 15 (of 27) key elements for a simple substitution, using bigram or trigram analysis alone.)

In order to carry out an investigation into the relative merits of different optimisation techniques for cryptanalysis, the tool shall be engineered to support a variety of metaheuristic algorithms as interchangeable components. Since many well-known metaheuristics rely solely on the *mutation* and *crossover* operators, it is necessary to define versions of these operators that are compatible with each type of key structure.

#### Mutation Operator

The mutation operator is an essential component of many local[5] and global optimisation techniques. To support the major classes of key structure – sequences and keywords – two different mutation operators are adopted, which are defined as follows:

**Swap mutation:** two elements in the sequence are selected and swapped. For example, a swap mutation on the sequence $\langle 1, 2, 3, 4 \rangle$ might yield the sequence $\langle 1, 4, 3, 2 \rangle$.

This operator does not require knowledge of the domain of the elements in the sequence. Furthermore, the elements themselves are not modified; if the sequence is a permutation, then swapping two elements is guaranteed to preserve the permutation property.

**Point mutation:** a letter in the keyword is selected and changed to another letter in the alphabet. For example, the keyword "`house`" may be mutated to "`ho`$\underline{\texttt{r}}$`se`" or "$\underline{\texttt{m}}$`ouse`".

This operator is better suited for keyword-type key structures than swap mutation, because the letters of the keyword can be changed individually, providing search with the potential to reach every possible $n$-letter keyword.

Many variations of these operators are possible: for instance, Jakobsen (1995) adapts swap mutation to exchange elements in close proximity to each other, in preference to elements that are located further apart. For the sake of generality, these variations are eschewed, but could easily be incorporated within the key structures if desired.

#### Crossover Operator

A crossover operator is required by some evolutionary metaheuristics, such as genetic algorithms. The role of crossover is to combine the elements of multiple parent sequences in some manner, whilst preserving some characteristics of each parent.

Two common forms of the crossover operation are "one-point" and "two-point" crossover. One-point crossover is generally regarded to be inferior to two-point crossover in most situations (Michalewicz, 1996). Many other methods of crossover have been proposed in the literature[6]; however, the two-point crossover operator is adopted here, due to its simplicity.

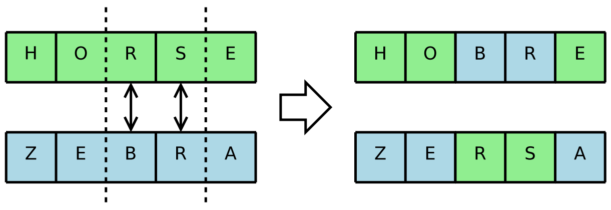


Figure 4.4: A visual representation of the two-point crossover operator. (The dotted lines mark the crossover points.)

[5]For the purposes of local search techniques, the neighbourhood $\mathcal{N}$ of a state $s$ is equivalent to the set of states which may be obtained by executing a single mutation on $s$.

[6]These operators include – but are not limited to – multi-point, adaptive, segmented, shuffle and uniform crossover (Michalewicz, 1996).

To perform two-point crossover, a pair of crossover points are selected, then the blocks of elements in each parent sequence that lie between the crossover points are exchanged. As shown in Figure 4.4, the resulting child sequences bear a resemblance to each of their parents.

Care must be taken when performing crossover on permutation sequences, since elements of parents cannot be exchanged indiscriminately whilst maintaining the permutation property in the children. In order to resolve this problem, the crossover operator must be revised.

An adaption of two-point crossover that respects the permutation property is *partially-mapped crossover* (PMX). There are many small variations in the definition of the PMX operator in the literature (Eiben and Smith, 2003) and, unfortunately, most of these variants are poorly documented. The version of PMX described here is taken from Michalewicz (1996).

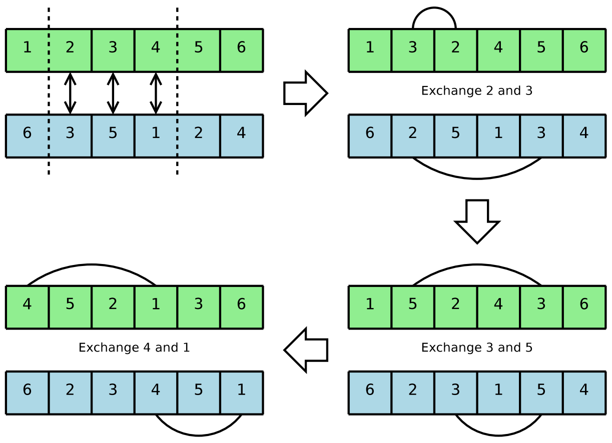


Figure 4.5: A step-by-step visual representation of the PMX operator.

The elements that lie between the crossover points define a series of mappings between pairs of elements: in the example shown in Figure 4.5, these are $2 \leftrightarrow 3$, $3 \leftrightarrow 5$ and $4 \leftrightarrow 1$. To execute PMX on a sequence, each mapping pair is applied in turn: that is, the two elements corresponding to the mapping pair are located in the sequence and swapped. The resulting children thus inherit characteristics of their parents, but retain the permutation property.

## 4.4 Control Layer

The control layer provides two "front-end" interfaces for accessing the facilities of the cipher and cryptanalysis layers:

**User interface:** a user-accessible suite for carrying out the following activities:

- constructing custom ciphers from pre-defined text functions.
- encrypting and decrypting arbitrary texts according to specified ciphers and keys.
- running (and monitoring) automated cryptanalysis on user-provided ciphertexts.

**Batch interface:** to supervise cryptanalysis runs without user intervention. This interface is intended for experimental purposes and to evaluate the capabilities of the tool.

The control logic serves to bridge the "front-end" interfaces with the "back-end" cipher and cryptanalysis layers. To avoid the unnecessary duplication of code, it supplies the functionality common to both interfaces.

# 5 Implementation and Testing

## 5.1 Choice of Development Language

The following criteria were considered to be significant towards the choice of development language for implementing the cryptanalysis tool:

**Maturity:** established languages are usually well-documented and, in addition, many boast sophisticated third-party development tools to assist the software development process.

**Object orientation:** is a powerful programming paradigm, which elevates the conceptual model of a program to a higher level of abstraction than that offered by procedural languages. Object-oriented (O-O) languages remove many of the *"accidental difficulties"* (Brooks, 1995) of programming, as they deliver modularity (by encapsulation) and re-usability (by polymorphism and inheritance).

**Run-time performance:** metaheuristic search is a computationally intensive process and will constitute the bulk of processing performed by the tool. To maximise the tool's run-time performance, the program code should be compiled to native machine code.

**Programmer expertise:** for any new software project, choosing an implementation language in which the developers have little (or no) previous experience carries an element of risk, because substantial amounts of time and resources must be invested in acquiring familiarity with the language. It is therefore advisable to select a language in which one is already competent, unless sound reasons can be given for doing otherwise.

**Integration with existing software:** as stated in Section 3.4, the reuse of existing software libraries in the tool is considered beneficial to the development process. To simplify the task of interfacing the tool with these libraries, the tool development language should be compatible with existing library code.

The author has experience with numerous O-O development languages, but claims expertise only with Java and Python, having acquired familiarity with these languages in previous programming projects. Both Java and Python are considered to be mature languages; as such, either would be suitable for implementing the tool.

It is well known that a program compiled to native machine code will yield considerable run-time performance gains over an equivalent interpreted program. Early versions of the Sun Java compiler[1] were criticised for poor performance, but this deficiency has been corrected in more recent versions of the Java compiler, to the extent that *"there is rarely any significant performance difference between native C++ and Java applications"* (Mangione, 1998).

However, the same claims cannot be made for the mainstream Python implementation, which is a byte-code interpreter, rather than a compiler. Consequently, Java programs can usually be executed an order of magnitude faster than interpreted Python code (Nystrom, 2007). The relatively poor performance of Python is considered a significant drawback to its adoption for this project, due to the aforementioned processing requirements of metaheuristic algorithms. Hence, Java is selected as the implementation language for the tool software.

[1]Java code is typically pre-compiled to Java virtual machine (JVM) byte-code and then translated to native machine code at run-time, in a process known as "just-in-time" (JIT) compilation. For the purposes of this project, Java is considered a compiled language.

## 5.2 Choice of Development Tools and Software Libraries

The following freely-available software development tools were selected for the purposes of implementing the cryptanalysis tool:

**Development environment:** an integrated development environment (IDE) is a suite of applications designed to assist the software development process. Judicious use of the facilities of an IDE will increase programmer productivity and thus enhance the quality of the resulting software product.

**Eclipse** is an extensible (and highly customisable) IDE with comprehensive support for Java development. In addition, Eclipse features a sizable library of "plug-in" tools, which are targeted to various aspects of the development process.

**Unit testing:** in keeping with the extreme programming method (Section 3.3), a suite of unit tests will be created to validate individual modules. As part of regression testing, the unit tests will be re-run after each modification to the code-base, to catch any unintended regressions in program functionality.

The *de facto* standard for unit testing in Java is the **JUnit** framework, which is therefore adopted for the project.

**Version control:** a version control system (VCS) is a tool to manage and track changes to source code. Whilst the primary purpose of a VCS is to synchronise a code-base between multiple developers, it is still advantageous to deploy a VCS in a single-person project, to maintain backups of the code-base and, should the need arise, to restore earlier versions of modules.

The VCS selected for the software development is **Subversion**. (This choice is made purely on the basis of the author's personal experience with Subversion; many other VCS tools would be equally suitable for the purposes of the project.)

Furthermore, the ECJ and SWT software libraries are used to implement major areas of tool functionality, as described in Subsection 5.3.5 and Subsection 5.3.6 respectively.

## 5.3 Implementation Details

### 5.3.1 Text Services

The conceptual text model outlined in Subsection 4.2.1 is reified to provide the tool with functionality to store and manipulate plaintexts and ciphertexts.

The *Symbol*, *Letter* and *Alphabet* classes are direct mappings to their counterparts in the conceptual model. These classes are immutable: their contents are fixed at initialisation and cannot be changed subsequently.

A *Message* encapsulates a private list of *Symbol* (and *Letter*[2]) objects; individual symbols may be retrieved or changed with the `getSymbol` and `setSymbol` operations. (The methods `toString` and `setText` are required by higher-level modules to support I/O operations on messages, such as loading messages from text files.)

An alternative implementation of the text model would use Java's built-in `char` and `string` types to directly represent message texts. This approach would probably be more efficient (in terms of memory usage) than the chosen implementation, but lacks the flexibility to define custom alphabets and, hence, cannot capture the distinction made between letters and non-letters in Subsection 4.2.1.

[2]Since a letter is a special type of symbol, the *Letter* class inherits the properties of *Symbol*.

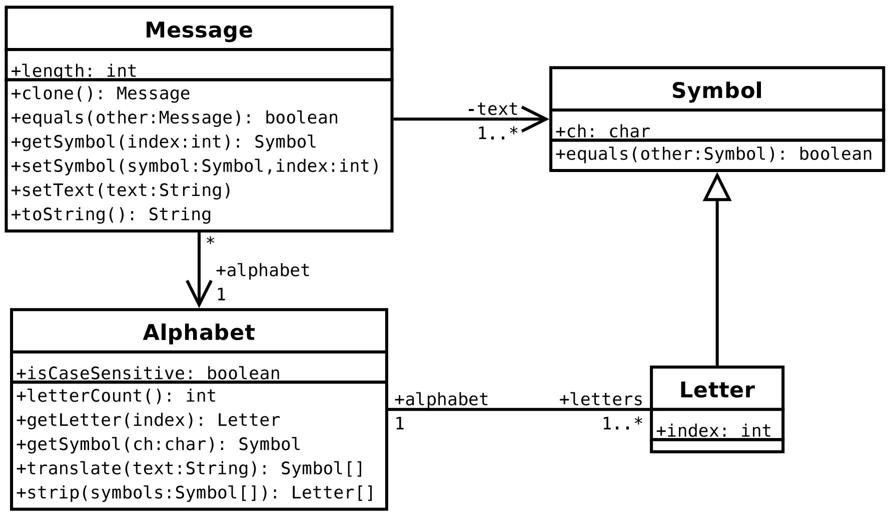

Message
+length: int
+clone(): Message
+equals(other:Message): boolean
+getSymbol(index:int): Symbol
+setSymbol(symbol:Symbol,index:int)
+setText(text:String)
+toString(): String
-text
1..*
Symbol
+ch: char
+equals(other:Symbol): boolean
*
+alphabet
1
Alphabet
+isCaseSensitive: boolean
+letterCount(): int
+getLetter(index): Letter
+getSymbol(ch:char): Symbol
+translate(text:String): Symbol[]
+strip(symbols:Symbol[]): Letter[]
+alphabet
1
+letters
1..*
Letter
+index: int

Figure 5.1: A UML representation of the text classes, which builds on Figure 4.2.

### 5.3.2 Ciphers and Functions

The *Cipher* class implements a "cipher pipeline" of *Function* objects. When an instance of *Cipher* is called to encrypt or decrypt a *Message* object, a copy[3] of the message is passed through each function in the cipher pipeline in sequence (as described in Subsection 4.2.2). The transformed message is then returned.

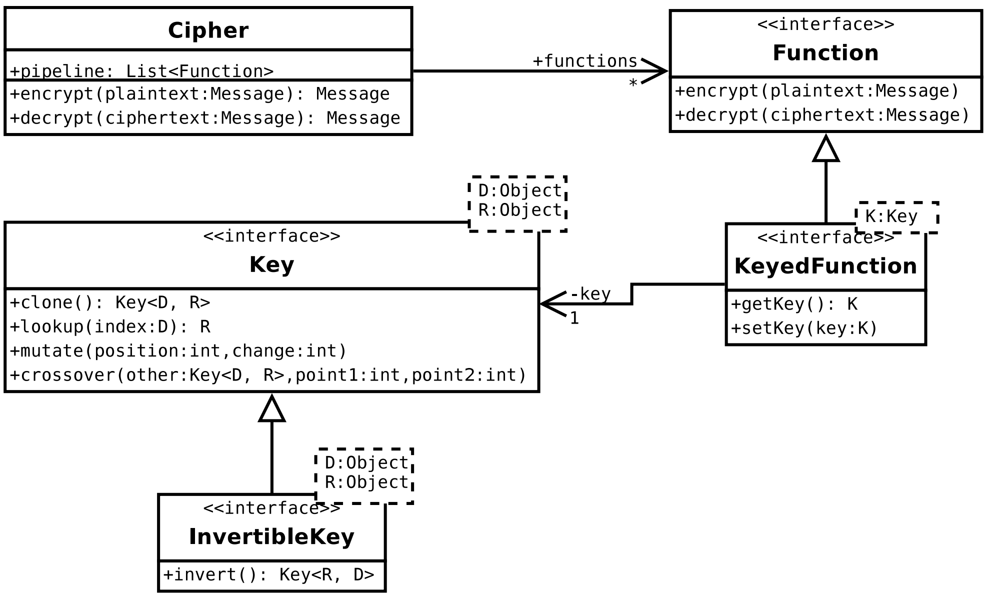

Cipher
+pipeline: List<Function>
+encrypt(plaintext:Message): Message
+decrypt(ciphertext:Message): Message
+functions
*
<<interface>>
Function
+encrypt(plaintext:Message)
+decrypt(ciphertext:Message)
D:Object
R:Object
<<interface>>
Key
+clone(): Key<D, R>
+lookup(index:D): R
+mutate(position:int,change:int)
+crossover(other:Key<D, R>,point1:int,point2:int)
-key
1
K:Key
<<interface>>
KeyedFunction
+getKey(): K
+setKey(key:K)
D:Object
R:Object
<<interface>>
InvertibleKey
+invert(): Key<R, D>

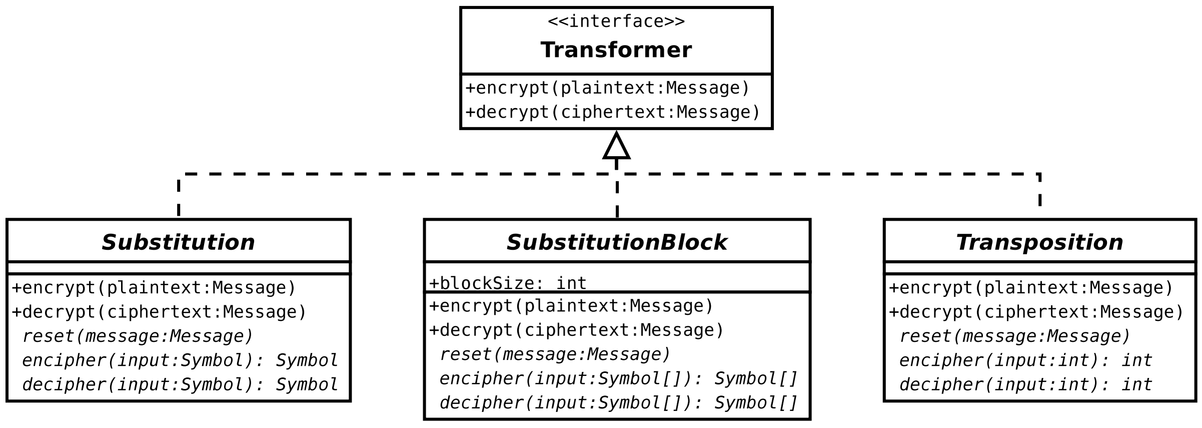
Figure 5.2: A UML representation of the components in the cipher model.

The `encrypt` and `decrypt` operations of the *Function* class perform in-place transformation on *Message* objects: the symbols of the message text are overwritten with the transformed symbols. This avoids the overhead of initialising a new *Message* object for each function in the pipeline, thus improving run-time efficiency.

Text functions may be classified as being either substitutions or transpositions. All substitution functions operate according to the same principle – as do all transposition functions – so the common characteristics of each type of function are implemented separately as (abstract)

[3]The *Message* is copied to ensure that the original text is still available after encryption or decryption.

classes conforming to the *Transformer* interface (Figure 5.3). This approach promotes reusability and avoids the unnecessary duplication of code in each *Function* implementation.



<<interface>>
Transformer
+encrypt(plaintext:Message)
+decrypt(ciphertext:Message)
Substitution
+encrypt(plaintext:Message)
+decrypt(ciphertext:Message)
reset(message:Message)
encipher(input:Symbol): Symbol
decipher(input:Symbol): Symbol
SubstitutionBlock
+blockSize: int
+encrypt(plaintext:Message)
+decrypt(ciphertext:Message)
reset(message:Message)
encipher(input:Symbol[]): Symbol[]
decipher(input:Symbol[]): Symbol[]
Transposition
+encrypt(plaintext:Message)
+decrypt(ciphertext:Message)
reset(message:Message)
encipher(input:int): int
decipher(input:int): int

Figure 5.3: The trio of implementations of the *Transformer* interface.

Each *Transformer* class provides a generic implementation of the `encrypt` and `decrypt` methods. This reduces a *Function* definition to specifying the `encipher` and `decipher` methods required by the transformer:

- The *Substitution* transformer operates on a "symbol-by-symbol" basis. When the `encrypt` method is called with a *Message* as its argument, each individual letter in the message is extracted and passed to the `encipher` method: this returns the equivalent ciphertext letter, which replaces the original letter in the message.

  Substitution functions which operate on fixed-size blocks of symbols (polygraphic substitution) extend the *SubstitutionBlock* transformer, whose `encipher` and `decipher` methods map between symbol blocks in a similar manner.

- The *Transposition* transformer works in terms of the index positions of symbols. To encrypt a message of length $N$, each symbol index in the range $1..N$ is passed into the `encipher` method, which returns the corresponding index of the symbol in the ciphertext. The symbols of the message are then reordered accordingly, with the assistance of a buffer to hold temporarily-displaced symbols.

Whilst all functions must conform to the *Function* interface, each function class extends an implementation of the *Transformer* interface – *Substitution*, *SubstitutionBlock* and *Transposition*. The relationship of several *Function* classes to these transformers is shown in Figure 5.4.

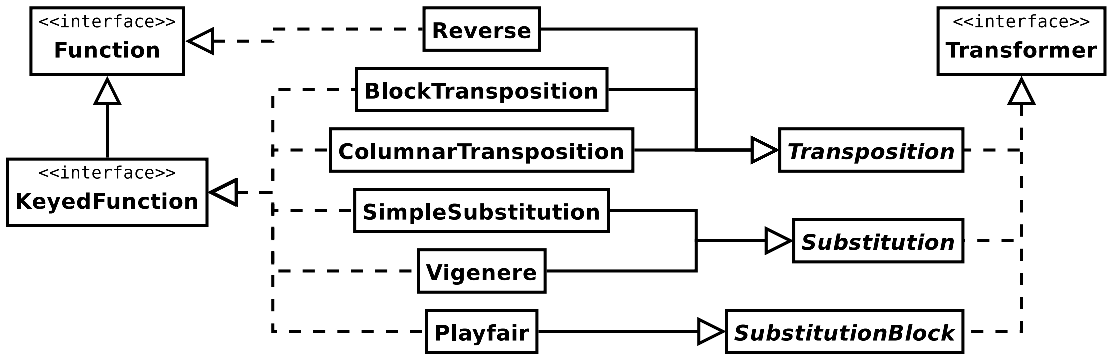


Figure 5.4: A taxonomy of *Function* classes implemented for the tool.

Functions that require a key parameter are distinguished by the *KeyedFunction* interface. Since there are different types of keys, the type of *Key* object required by a *KeyedFunction* must be specified as a class parameter by the function declaration.

### 5.3.3 Key Structures

The *Key* interface defines a set of operations for accessing the key data that is stored within a key structure. Hence, each of the multiple key structures identified in Subsection 4.2.2 corresponds to a *Key* implementation.

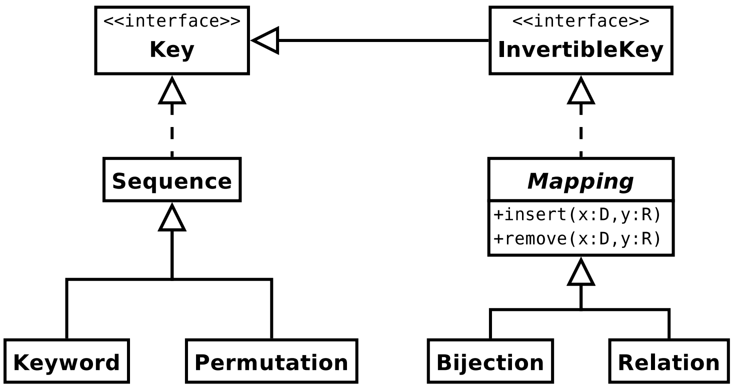


Figure 5.5: The hierarchy of *Key* classes.

Since many of the key structures share a similar design, their implementations are organised in a class hierarchy, as shown in Figure 5.5. Two base "styles" of key structure are identified:

**Mappings:** The *Mapping* class[4] represents a relation between a "domain" set of discrete elements and a "range" set. Internally, the map structure is implemented as a hash table, which permits retrieval of elements in $O(1)$ time (Cormen et al., 2001).

Substitution keys that map a plaintext alphabet to a ciphertext alphabet are implemented as subclasses of *Mapping*. The *Bijection* class enforces a one-to-one mapping – as required by simple substitution – whereas the *Relation* class allows elements to be mapped to sets of elements (and is, therefore, suitable for homophonic substitution).

The *Mapping* class provides an `invert` method, since the decryption key may be obtained by simply inverting the encryption mapping (and vice versa).

**Sequences:** The *Sequence* class (and subclasses) represent keys whose elements are indexed by their "position" in the key, as is commonplace for transposition ciphers.

The *Keyword* class is specialised for sequences of *Letter* objects by using point mutation (Subsection 4.3.2) and serves as the key for various polyalphabetic substitution functions.

The `mutate` and `crossover` methods implement the evolutionary operators described in Subsection 4.3.2. These methods are deterministic: each accepts two integer values, which are internally translated to the parameters for the operation to be carried out[5]. (These values are generated in the cryptanalysis layer.)

[4]In mathematics, the term "mapping" is synonymous with "function"; however, the former word is used here, to avoid confusion with the *Function* class.

[5]To perform PMX crossover on *Mapping* keys, the range elements are treated as a permutation, with the domain elements as the corresponding indexes.

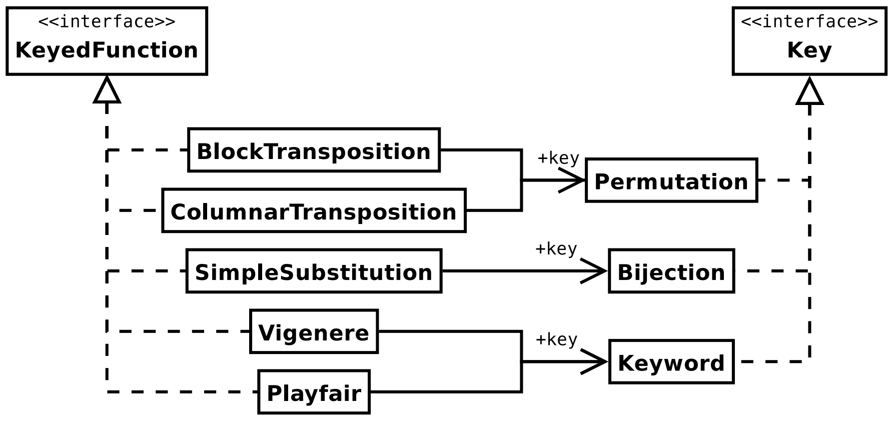


Figure 5.6: The relationships between several *Function* and *Key* implementations.

### 5.3.4 Evaluation of Decrypts

A *MessageEvaluator* measures the degree of resemblance between a decrypted *Message* text and the characteristics of natural language; this evaluation is expressed as a numerical score. Hence, an implementation of *MessageEvaluator* is analogous to a fitness measure.

A collection of *MessageEvaluator* instances (with associated weight values) may be grouped together by a *Combiner* class – itself an implementation of *MessageEvaluator* – which simply applies each evaluator to a decrypt and sums the weighted scores.

In addition to *Combiner*, two *MessageEvaluator* classes are implemented:

- The *FrequencyAnalysis* evaluator calculates the normalised $n$-gram difference statistic for a message text. The data sources of the "expected" $n$-gram frequencies are separated by gram size; hence, to model the unigram, bigram and trigram components of the fitness function, three separate *FrequencyAnalysis* instances are created.

- The *WordMatcher* evaluator calculates the "dictionary heuristic" score (Subsection 2.3.3) for a message text. (A machine-readable word list[6] is used as the dictionary source.)

  Upon its initialisation, the *WordMatcher* constructs a "trie" data structure (Figure 5.8) to store the dictionary of words in a compact format which permits linear-time lookup of individual words. To evaluate a *Message*, the Aho and Corasick (1975) string-matching algorithm[7] is applied (together with the trie of words) to extract the words present in the message text in a single pass.

The fitness function described in Subsection 4.3.1 is realised as a *Combiner* containing instances of the *FrequencyAnalysis* and *WordMatcher* evaluators. A *CombinerFactory* class is used to generate *Combiner* instances that correspond to a user-provided *CombinerParameters* object, which simply holds a set of weight values.

The flexibility of this model allows the user to define fitness functions tailored to different styles of ciphertext: for example, the *FrequencyAnalysis* class may be initialised with a corpus of $n$-gram statistics suitable for a particular language, or an additional *WordMatcher* instance may introduced to recognise specialist vocabulary in decrypted texts.

[6]The "SCOWL 5desk" English word list is available at http://wordlist.sourceforge.net/

[7]The choice of an established algorithm – as opposed to devising a custom string-matching algorithm – reflects the preference for reuse of existing techniques, as stated in Section 3.4.

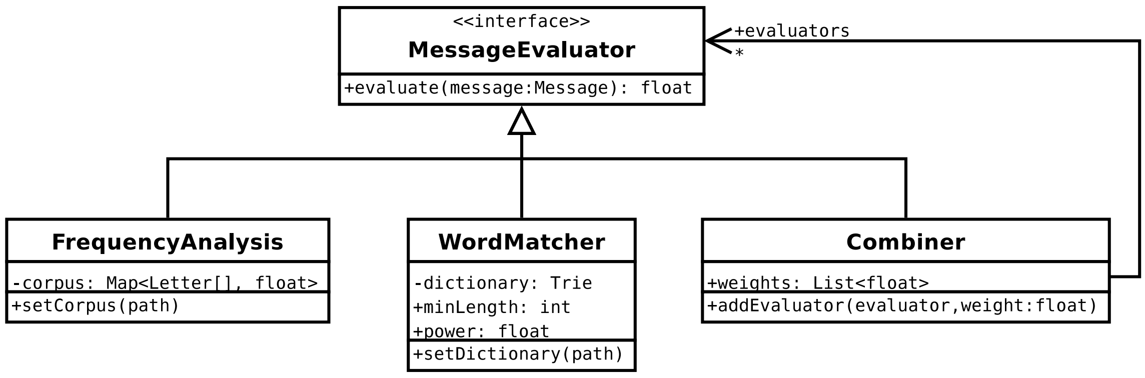


Figure 5.7: The implementations of the *MessageEvaluator* interface.

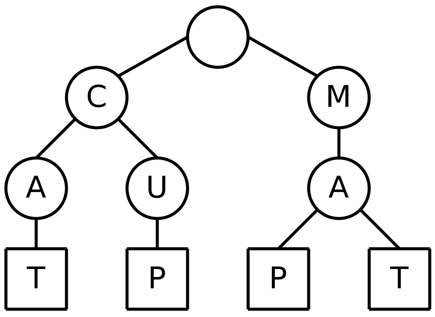


Figure 5.8: An example "trie" data structure, containing the strings 'cat', 'cup', 'map' and 'mat'. (Word terminals are denoted by square nodes; non-terminals by circles.)

### 5.3.5 Cryptanalysis Engine

#### Existing Implementations of Optimisation Algorithms

It was stated in Subsection 4.3.2 that the cryptanalysis layer of the tool should support a selection of metaheuristic search algorithms. To save time and resources, existing implementations of these algorithms are adapted and integrated with the tool, instead of coding each individual algorithm from scratch.

The open-source Evolutionary Computation for Java (ECJ) toolkit (Luke, 2007) includes "off-the-shelf" implementations of many evolutionary metaheuristics, including genetic algorithms and evolutionary strategies. ECJ is *"designed to be highly flexible"*: almost all functionality in the toolkit can be extended (or overridden) to accommodate any special requirements (McIntyre et al., 2004). Unfortunately, the documentation[8] for ECJ is rather sparse, but the author has acquired experience of using ECJ in a previous project.

The design of ECJ is targeted to population-based metaheuristics and, consequently, ECJ does not support local search methods. Hence, a generic Java implementation[9] of simulated annealing (with minor modifications) was used to perform experiments with local search.

#### Searching the Keyspace

The *Cracker* class (Figure 5.9) is the central facility for coordinating the cryptanalysis of ciphertexts. This helps to keep the cryptanalysis and control layers separate, by abstracting (hiding) the implementation details of the cryptanalysis layer from the control layer.

[8]Documentation for ECJ is available at http://www.cs.gmu.edu/~eclab/projects/ecj/docs/
[9]Mark Ramirez's BSD-licenced implementation of simulated annealing: http://www.mrami.com/~mrami/public/

Before a *Cracker* instance can commence the cryptanalysis of a ciphertext, the ciphertext itself must be specified (by setting the appropriate field in *Cracker*), together with an initialised *Cipher* object, in order to obtain candidate decryptions of the ciphertext. Other parameters, such as the choice of optimisation algorithm and the respective weights of each component of the fitness function, may also be specified.

The *Cracker* class implements the Java *Runnable* interface, allowing a cryptanalysis run to be executed in parallel with other parts of the tool. The progress of a running *Cracker* can be monitored by subscribing to the cracker's associated *EventRelay* object, which provides event notification in the style of the standard "observer" design pattern (Gamma et al., 1995).

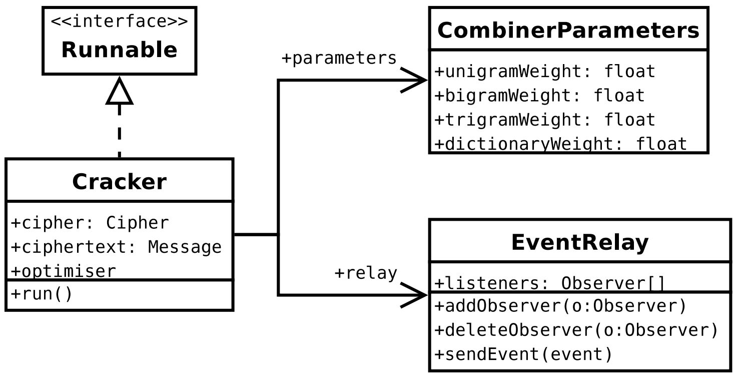


Figure 5.9: The *Cracker* class and associated classes.

Since the optimisation algorithms of ECJ expect the fitness function to yield higher scores for "better" decryptions, the raw fitness evaluation of a candidate key (as given by the cost function $f$) is adjusted to a value in the closed interval $[0, 1]$ according to the formula $Adj_f = \frac{1}{1+f}$.

The *KeyFactory* class provides an "abstract factory" service (Gamma et al., 1995) to generate randomised *Key* objects and, as such, is employed by population-based optimisation algorithms to create an initial population of candidate keys (or, in the case of local search, the initial "key state"). To cater for the multiple *Key* implementations, several *KeyFactory* subclasses are needed, as shown in Figure 5.10.

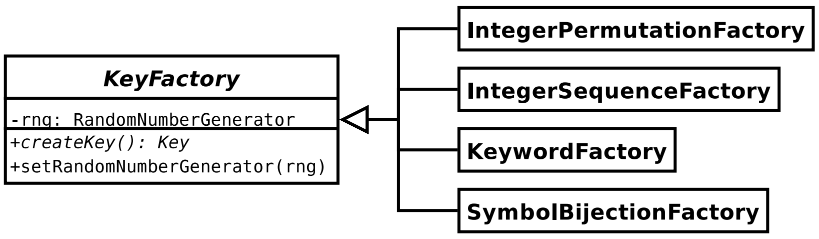


Figure 5.10: The *KeyFactory* class and its subclasses.

### 5.3.6 User Interface

#### Choice of User Interface Toolkit

At present, the two most common types of user interface are the graphical user interface (GUI) and the web-based user interface (accessed through a web browser). Since it is expected that the cryptanalysis tool will be used on a local workstation, the GUI model is preferred.

A GUI toolkit is a set of reusable graphical elements ("widgets") for the construction of user interfaces. There are two mainstream GUI toolkits available for Java, which are as follows:

**Swing** is a platform-independent GUI framework based on the model-view-controller (MVC) pattern. The "look and feel" of Swing widgets is independent of the host platform.

**Standard Widget Toolkit (SWT)** was conceived by the Eclipse project as an alternative to Swing. The SWT widget library is a portable interface to the GUI widgets of the underlying desktop environment: this enables SWT applications to take the "native" appearance of the host platform.

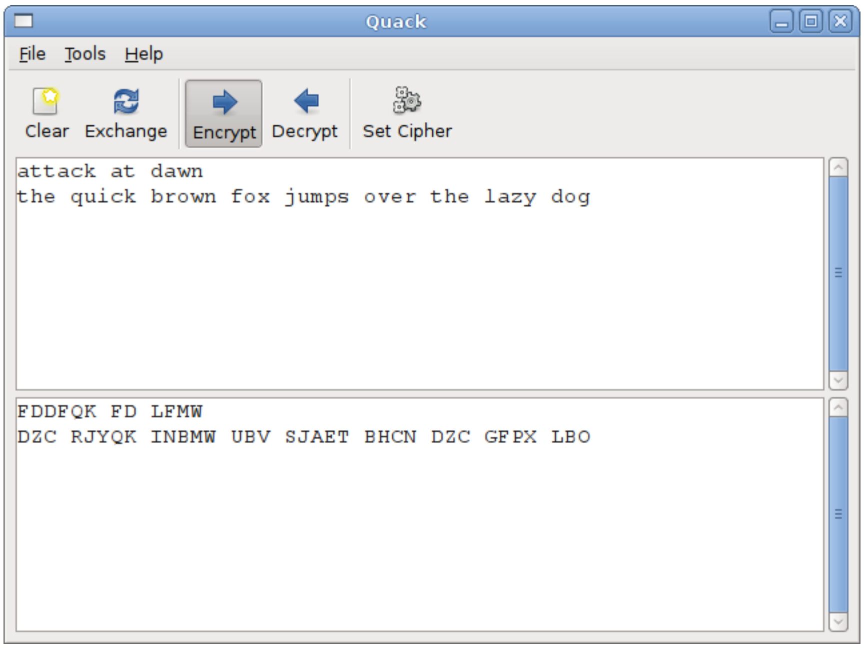
The relative technical merits of Swing and SWT are a matter of on-going debate[10]. However, it is generally accepted that SWT is more resource-efficient than Swing and, furthermore, the SWT API is regarded as being somewhat easier to learn than the MVC architecture of Swing (Scarpino et al., 2004). For these reasons, SWT was selected as the GUI toolkit for implementing the tool front-end.

### Design of the User Interface

*Screenshots of all the windows displayed by the tool are located in Appendix B.*

The user interface is split across multiple windows. Each window provides access to a portion of the underlying functionality of the tool. Upon starting the tool, the main window (Figure 5.11) is presented to the user; the other windows can be accessed from the "Tools" menu displayed at the top of the main window.

The main window is divided into two panes: the upper "source" pane contains a user-editable text display, with the corresponding encrypted (or decrypted) text displayed in the (read-only) lower "view" pane. An ASCII-formatted text file may be loaded from disk to be displayed in the source pane; similarly, the contents of the view pane can be saved to a file.



```
attack at dawn
the quick brown fox jumps over the lazy dog
```

```
FDDFQK FD LFMW
DZC RJYQK INBMW UBV SJAET BHCN DZC GFPX LBO
```

Figure 5.11: The main window of the tool's user interface.

The user can select the cipher to be used from a drop-down list of pre-defined ciphers, which is accessed by clicking the "Set Cipher" button. Alternatively, the "Cipher Constructor" dialog window (Figure B.1) may be invoked to define a custom cipher pipeline. Functions may

[10] A discussion of the different "design philosophies" of Swing and SWT is given by Scarpino et al. (2004).

be added, removed or reordered in the pipeline and, by selecting a function, the key parameter for the function is displayed (and can be changed).

The cryptanalysis facilities of the tool are accessible from the "Cracker Configuration" window (Figure B.2), from which the weights of each component of the fitness function may be set. Cryptanalysis is started by clicking the "Run Cracker" button, upon which the "Cracker Monitor" window (Figure B.3) is launched, to display the progress of the cryptanalysis run.

## 5.4 Summary of Software Testing

As befitting the agile methodology of software development, the following testing activities were undertaken in parallel with the tool's construction:

**Unit testing:** a suite of unit tests was written to cover each class in the cipher and cryptanalysis layers. (The control layer was omitted from unit testing, because it is generally accepted that user interfaces do not lend themselves well to automated testing practices.)

Naturally, unit tests written for higher-level components must assume the correctness of lower-level modules. This is acceptable, since those lower-level modules will already have been extensively unit tested.

**Integration testing:** each layer of the tool (Subsection 4.1.2) was constructed and tested separately, in a "bottom-up" fashion: that is, the cipher layer was built first, followed by the cryptanalysis layer and, finally, the control layer.

It should be stated that integration testing revealed some flaws in the interfaces between classes, which required rectification. (The implementation described in Section 5.3 reflects the final version of the tool.)

**System testing:** was performed by applying the completed tool to a set of sample ciphertexts, in order to identify (and resolve) any deficiencies present in the tool's construction that were overlooked in unit and integration testing.

The process of system testing is somewhat related to evaluating the tool's capabilities; hence, the experimental work reported in Chapter 6 may be viewed as a continuation of system testing.

# 6 Experimentation with the Tool

*This chapter explores the capabilities of the completed cryptanalysis tool. Suitable weights for the fitness function are identified by experimenting with a simple substitution cipher. The efficacy of the tool is investigated by attacking a variety of classical ciphers. Attempts are made to cryptanalyse the unsolved cryptograms described in Section 1.4.*

## 6.1 Choice of Fitness Function

Following on from the discussion in Subsection 4.3.1, the fitness function is a weighted combination of the unigram, bigram, trigram and dictionary heuristic measures:

$$Cost(k) = \alpha \cdot uni(k) + \beta \cdot bi(k) + \gamma \cdot tri(k) - \delta \cdot dict(k)$$

For short ciphertexts, only a small fraction of all possible bigrams and trigrams will occur in a decryption. Hence, an important optimisation[1] is introduced for the $n$-gram measures: only those $n$-grams which occur in a decrypted text are counted and compared with the reference $n$-gram frequencies, instead of performing comparisons for all possible $n$-grams.

This practice reduces the computational complexity of $n$-gram analysis – from polynomial-time (in $n$) to linear-time (in the amount of ciphertext) – and significantly improves the run-time performance for evaluating a decryption (especially for trigram analysis). The accuracy of the fitness function is not adversely affected, since the penalty incurred by a decryption for failing to match the omitted $n$-grams is intuitively reflected in the disparity between the frequencies of the $n$-grams present in the decryption and the respective expected frequencies.

## 6.2 Calibrating the Evaluation Measures

Since each evaluation measure operates independently of the others, it is necessary to scale the scores produced by each measure, so that, when applied to a decrypted text, equally-weighted measures will produce roughly equal scores.

A selection of 10 English literary works (non-copyrighted, or copyright expired) were sourced from the Project Gutenberg[2] catalogue, to be used as natural language "training" material. Each chosen text is written in a different style of language, which leads to variations in the statistical profiles of the texts.

The unigram, bigram, trigram[3] and dictionary measures were applied individually to evaluate each text in the training set. The results (reported in Table 6.1) indicate that each measure produces reasonably consistent evaluations across the training set, as evidenced by the low standard deviations of the scores. Notably, the variations between texts are insufficient to distort their statistical profiles beyond what is recognisable as natural language text.

In the following experiments, the raw scores produced by each evaluation measure for each decryption are divided by the measure's mean score from Table 6.1, to scale the evaluation measures to a common range. This ensures that no measure has a disproportionate effect upon the combined fitness score of a decryption.

[1]This technique is not described in any paper reviewed in Chapter 2 and therefore appears to be original.
[2]The online Project Gutenberg book library is available at http://www.gutenberg.org/catalog/
[3]These measures were initialised with $n$-gram statistics extracted from an unrelated corpus of English texts.

| | Scores of Evaluation Measures | | | |
|---|---|---|---|---|
| **Title of Text** | **Unigram** | **Bigram** | **Trigram** | **Dictionary** |
| *20,000 Leagues Under the Sea* † | 4.73 | 212.87 | 1967.77 | 3.02 |
| *Alice's Adventures in Wonderland* | 6.79 | 232.14 | 2026.79 | 2.17 |
| *Dr. Jekyll and Mr. Hyde* | 6.40 | 248.25 | 1900.19 | 2.69 |
| *Hamlet* | 5.62 | 255.11 | 2057.49 | 2.24 |
| *Oliver Twist* | 6.50 | 254.48 | 2166.17 | 2.62 |
| *Pride and Prejudice* | 5.77 | 249.32 | 1953.47 | 3.05 |
| *Robinson Crusoe* | 6.61 | 256.40 | 1985.34 | 2.41 |
| *The Adventures of Tom Sawyer* | 8.21 | 262.41 | 2134.29 | 2.46 |
| *The Prince* † | 4.46 | 204.82 | 1773.71 | 3.05 |
| *Ulysees* | 6.03 | 269.64 | 2460.74 | 2.72 |
| **Mean Evaluation Score** ($\mu$) | 6.11 | 244.54 | 2042.60 | 2.64 |
| **Scaled Standard Deviation** ($\sigma/\mu$) | 0.1754 | 0.0870 | 0.0906 | 0.1233 |

Table 6.1: Raw evaluation scores produced by each measure when applied to a sample set of 10 English literary texts. (The † denotes texts translated from their original language.)

## 6.3 Identifying Optimal Weights for the Fitness Function

For the cryptanalysis tool to achieve successful decryptions, it is necessary to "tune" the fitness function by identifying appropriate weight values for each evaluation measure. This is accomplished by trialling a range of values for a weight (the independent variable) over a set of sample ciphertexts; the quality of the resulting decryptions is the dependent variable.

### 6.3.1 Experimental Method

The sample plaintexts are the first chapters of each of the 10 literary works listed in Table 6.1, from which all punctuation and spacing is removed to harden the task of cryptanalysis[4]. The ciphertext material was obtained by encrypting the sample plaintexts with a simple substitution cipher, using the key shown in Figure 1.1.

Since the training material (the sample plaintexts) is known, the correctness of a decryption key is quantified by counting the number of elements in the key which are set correctly. This metric provides an objective measure of a decryption's quality and thus reflects the suitability of the corresponding weight value. Hence, the value found to consistently produce the most correct decryptions shall be selected as the optimal value for the weight.

A standard genetic algorithm was selected as the optimisation technique for this experiment. A set of GA control parameters suitable for cryptanalysis (Table 6.2) were identified by preliminary investigation of the tool during its development[5].

| | | | |
|---|---|---|---|
| Number of generations: | 50 | Population size: | 200 |
| Probability of mutation ($p_m$): | 0.4 | Selection method: | Tournament (size 12) |
| Probability of crossover ($p_c$): | 0.9 | Elite individuals: | 0 (no elitism) |

Table 6.2: Control parameters selected for the genetic algorithm.

The choice of tournament selection is expedient for the purposes of this experiment, as this selection method is insensitive[6] to the absolute values of the fitness scores. Therefore, it is only

[4]The deletion of non-letters from ciphertexts is standard practice in classical cryptography (Subsection 1.3.3).

[5]The high value of $p_m$ reflects the probability of a single mutation to each candidate key, instead of a mutation of each single element within each key.

[6]Tournament selection considers the rank of individuals in a population, rather than their relative fitness scores.

necessary to experiment with the ratios between weights, eliminating the need to investigate the magnitude of each individual weight.

Due to the randomisation properties inherent in metaheuristic search, multiple cryptanalysis runs will produce decryptions of varying quality. To compensate for the effects of these fluctuations, the cryptanalysis of each sample ciphertext is repeated 3 times[7] for each weight value. From these runs, the best decryption achieved (as measured by the number of correct letters) is selected as representative for the weight value; the other runs are disregarded[8].

### 6.3.2 Performance of Individual Measures

In order to determine the baseline performance of each $n$-gram measure, cryptanalysis runs were carried out using three fitness functions, consisting solely of unigram, bigram and trigram measures respectively.

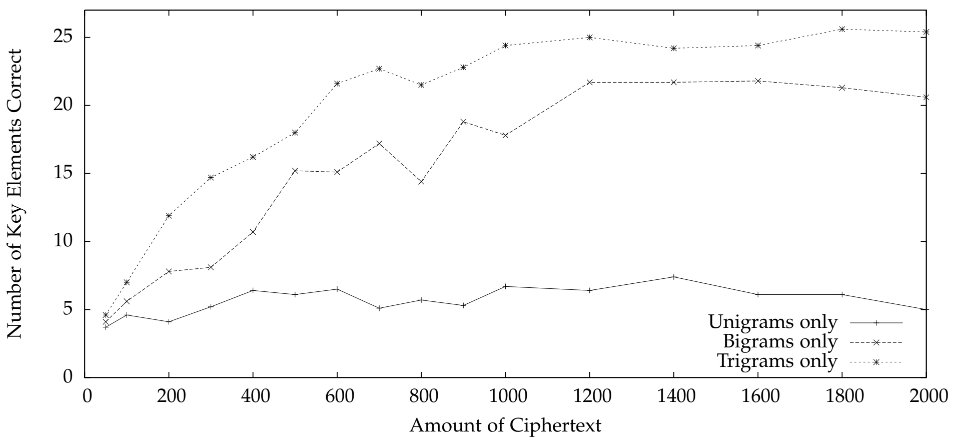


Figure 6.1: Performance of fitness functions comprising only a single $n$-gram measure.

The results obtained (Figure 6.1) correlate with those presented by Clark (1998) (reproduced in Figure 2.2, see page 22), which indicates that the tool is working correctly. However, there is some disparity between these two sets of results: in particular, the trigram-only fitness function exhibits significantly better performance than the bigram-only function, whereas this divergence is rather less pronounced in Figure 2.2. These differences may be attributed to factors such as the nature of the ciphertext material[9], the exact $n$-gram reference statistics and the choice of algorithm parameters.

For the bigram-only and trigram-only fitness functions, the correctness of decrypts improves for longer ciphertexts, as is expected for frequency analysis (Subsection 1.3.1). Contrary to expectations, the performance of the unigram-only fitness function remains flat regardless of the amount of available ciphertext, which implies that unigram statistics alone are unsuitable as a general heuristic for cryptanalysis.

### 6.3.3 Unigram and Bigram Weights

To determine whether there is any advantage to be gained by combining the unigram and bigram measures, fitness functions with positive values of $\alpha$ and $\beta$ are trialled and compared with the performance of the bigram-only fitness function. The test hypotheses as follows:

[7]The author is aware that a greater number of cryptanalysis runs for each ciphertext would increase the reliability of results. However, time and resource constraints precluded further processing.

[8]This method is closely related to the experimental approach described by Clark (1998).

[9]As stated in Subsection 6.3.1, spacing and punctuation are removed from the ciphertexts, whereas they are retained by Clark (1998).

**Null hypothesis** $H_0$**:** the combination of unigram and bigram measures is not an improvement on the bigram measure alone.

**Alternative hypothesis** $H_1$**:** the combination of unigram and bigram measures yields significantly better performance than the bigram measure alone.

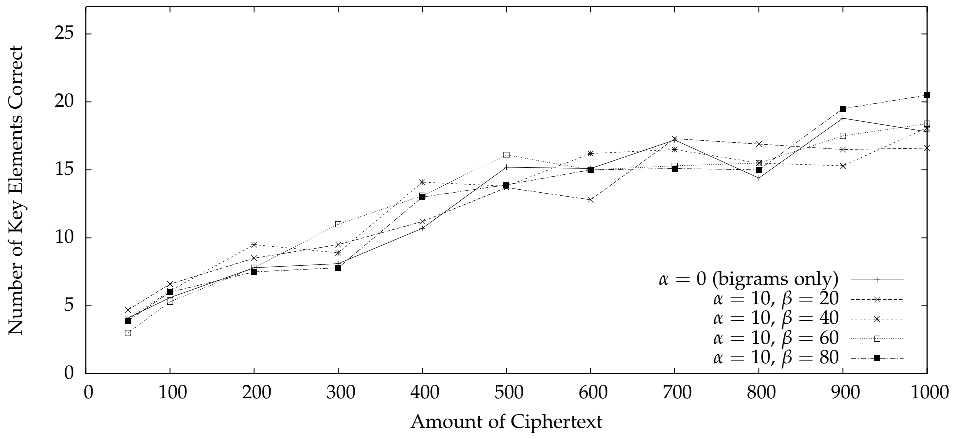


Figure 6.2: A comparison of the performance achieved by different values of $\beta$, respective to $\alpha$. (The bigram-only fitness function is included for reference.)

The probability distribution of these results is unknown and (without extensive analysis) cannot be presumed to follow a normal distribution. Hence, a non-parametric test statistic (which does not require assumptions regarding the probability distribution of a data set) must be used. Since the results are paired, the *Wilcoxon signed-rank test* is used, with tests conducted at the 5% significance level.

The test statistic is calculated by comparing the results obtained for each value of $\beta$ against those of the bigram-only fitness function. (As befitting the Wilcoxon signed-rank test, equal emphasis is placed on the performance scores for each increment of ciphertext.)

| | | **Amount of Ciphertext (letters)** | | | | | | | | | | **Test Statistic** | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\alpha$ | $\beta$ | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | 1000 | $p$ | $h$ |
| 0 | 10 | 5.6 | 7.8 | 8.1 | 10.7 | 15.2 | 15.1 | 17.2 | 14.4 | 18.8 | 17.8 | - | - |
| 10 | 10 | 4.8 | 8.4 | 10.2 | 12.5 | 14.2 | 14.5 | 16.4 | 15.6 | 16.6 | 16.3 | 0.8457 | 0 |
| 10 | 20 | 6.6 | 8.5 | 9.5 | 11.2 | 13.7 | 12.8 | 17.3 | 16.9 | 16.5 | 16.6 | 0.9219 | 0 |
| 10 | 40 | 6.1 | 9.5 | 8.9 | 14.1 | 13.8 | 16.2 | 16.5 | 15.5 | 15.3 | 18.1 | 0.4766 | 0 |
| 10 | 60 | 5.3 | 7.8 | 11.0 | 13.1 | 16.1 | 15.0 | 15.3 | 15.5 | 17.5 | 18.4 | 0.4961 | 0 |
| 10 | 80 | 6.0 | 7.5 | 7.8 | 13.0 | 13.9 | 15.0 | 15.1 | 15.0 | 19.5 | 20.5 | 0.5391 | 0 |

Table 6.3: Mean numbers of correct key elements identified for different values of $\alpha$ and $\beta$.

From Figure 6.2 and Table 6.3, it appears that high $\alpha : \beta$ ratios have no significant effect on the performance of cryptanalysis. This may be attributed to the "coarseness" of the unigram statistics.

Assigning greater importance to the bigram measure (with larger values of $\beta$) yields somewhat better performance, but does not surpass the performance of the bigram-only fitness function. Hence, the null hypothesis $H_0$ has not been disproved, so the unigram measure is omitted from the fitness function in subsequent experiments.

### 6.3.4 Bigram and Trigram Weights

As shown in Figure 6.1, both the bigram and trigram measures are independently capable of achieving good decryptions. A similar experiment is performed to determine whether these measures can be combined to yield better decryptions than trigram statistics alone. Again, the test hypotheses are as follows:

**Null hypothesis $H_0$:** the combination of bigram and trigram measures is not an improvement on the trigram measure alone.

**Alternative hypothesis $H_1$:** the combination of bigram and trigram measures yields significantly better performance than the trigram measure alone.

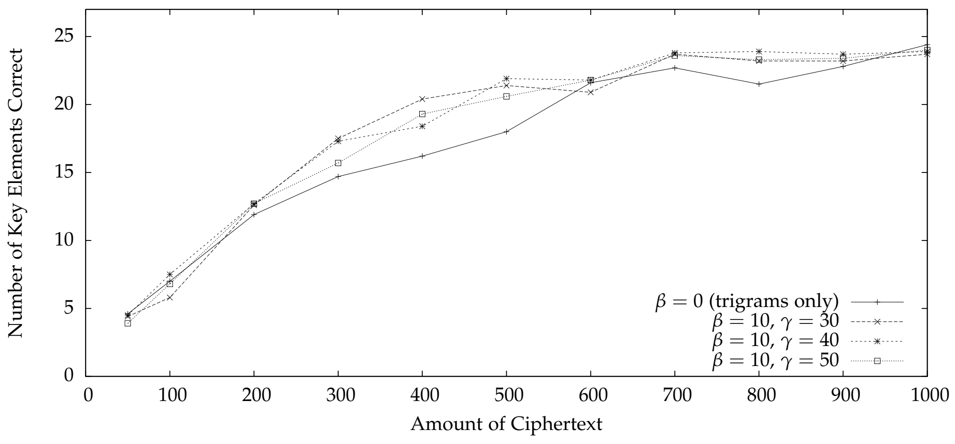


Figure 6.3: A comparison of the performance achieved by different values of $\gamma$, respective to $\beta$. (The trigram-only fitness function is included for reference.)

| | | Amount of Ciphertext (letters) | | | | | | | | | | Test Statistic | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\beta$ | $\gamma$ | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | 1000 | $p$ | $h$ |
| 0 | 10 | 7.0 | 11.9 | 14.7 | 16.2 | 18.0 | 21.6 | 22.7 | 21.5 | 22.8 | 24.4 | - | - |
| 10 | 10 | 6.2 | 8.6 | 14.6 | 18.5 | 18.8 | 20.3 | 20.5 | 20.8 | 21.5 | 22.7 | 0.1543 | 0 |
| 10 | 20 | 7.4 | 10.9 | 14.4 | 19.8 | 20.7 | 20.7 | 20.5 | 23.1 | 22.8 | 23.3 | 0.8203 | 0 |
| 10 | 30 | 5.8 | 12.6 | 17.5 | 20.4 | 21.4 | 20.9 | 23.7 | 23.2 | 23.2 | 23.7 | 0.1387 | 0 |
| 10 | 40 | 7.5 | 12.7 | 17.3 | 18.4 | 21.9 | 21.8 | 23.8 | 23.9 | 23.7 | 23.9 | **0.0078** | **1** |
| 10 | 50 | 6.8 | 12.7 | 15.7 | 19.3 | 20.6 | 21.8 | 23.6 | 23.3 | 23.4 | 24.0 | **0.0195** | **1** |

Table 6.4: Mean numbers of correct key elements identified for different values of $\gamma$.

As can be seen in Figure 6.3, the combinations of bigram and trigram measures generally enhance the performance of cryptanalysis, particularly with ciphertexts of fewer than 500 letters. The gain is maximised when $\gamma = 40$ and the corresponding test statistic – 0.0078 – is strong evidence that this result is not due to chance. Hence, $H_0$ is rejected in favour of $H_1$.

### 6.3.5 Dictionary Weight

The dictionary heuristic is intended to provide further refinement for a largely-correct decryption (Subsection 4.3.1). As before, the dictionary weight $\delta$ is adjusted with respect to $\beta = 10$ and $\gamma = 40$. The test hypotheses are as follows:

**Null hypothesis** $H_0$**:** the dictionary measure combined with the bigram and trigram measures is not an improvement on the combined bigram and trigram measures.

**Alternative hypothesis** $H_1$**:** the dictionary measure combined with the bigram and trigram measures yields significantly better performance than just the combined bigram and trigram measures.

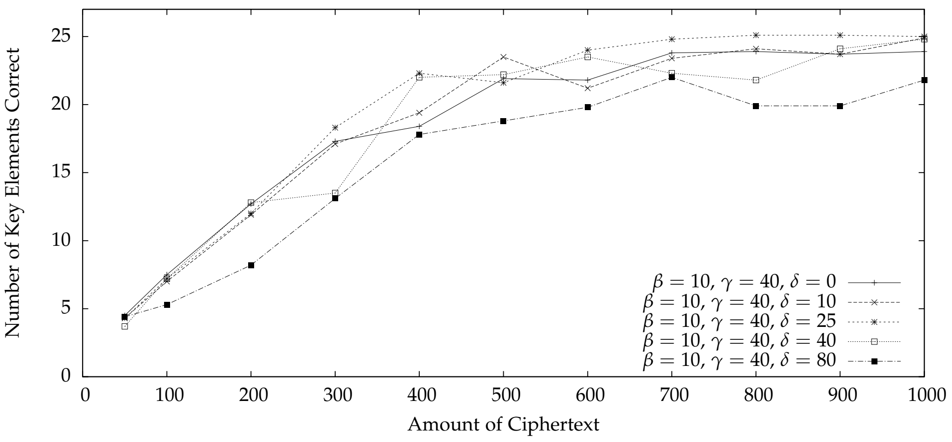


Figure 6.4: A comparison of the performance achieved by different $\beta : \gamma : \delta$ ratios.

| | | | **Amount of Ciphertext (letters)** | | | | | | | | | | **Test Statistic** | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\beta$ | $\gamma$ | $\delta$ | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | 1000 | $p$ | $h$ |
| 10 | 40 | 0 | 7.5 | 12.7 | 17.3 | 18.4 | 21.9 | 21.8 | 23.8 | 23.9 | 23.7 | 23.9 | - | - |
| 10 | 40 | 10 | 7.0 | 11.9 | 17.1 | 19.4 | 23.5 | 21.2 | 23.4 | 24.1 | 23.7 | 24.9 | 0.7070 | 0 |
| 10 | 40 | 20 | 6.6 | 9.4 | 19.8 | 21.6 | 23.0 | 23.8 | 22.5 | 22.8 | 24.9 | 25.6 | 0.3750 | 0 |
| 10 | 40 | 25 | 7.2 | 12.0 | 18.3 | 22.3 | 21.6 | 24.0 | 24.8 | 25.1 | 25.1 | 25.0 | **0.0254** | **1** |
| 10 | 40 | 30 | 6.8 | 17.1 | 17.4 | 20.6 | 24.8 | 23.5 | 23.6 | 24.4 | 23.5 | 22.9 | 0.2637 | 0 |
| 10 | 40 | 40 | 7.2 | 12.8 | 13.5 | 22.0 | 22.2 | 23.5 | 22.3 | 21.8 | 24.1 | 24.8 | 0.9219 | 0 |
| 10 | 40 | 60 | 5.6 | 13.0 | 15.0 | 20.7 | 18.6 | 20.8 | 22.5 | 22.6 | 23.2 | 22.9 | 0.0703 | 0 |
| 10 | 40 | 80 | 5.3 | 8.2 | 13.1 | 17.8 | 18.8 | 19.8 | 22.0 | 19.9 | 19.9 | 21.8 | 0.0020 | 0 |

Table 6.5: Mean numbers of correct key elements identified for different values of $\delta$.

The benefit of the dictionary heuristic becomes apparent above a threshold amount of ciphertext, whereupon a significant increase in the number of correctly-identified key elements can be observed (Figure 6.4). However, for shorter ciphertexts, values of $\delta$ exceeding 40 tend to unbalance the search[10], since the first dictionary words detected in a decryption will not necessarily be correct (Subsection 2.3.3).

The optimal value of $\delta$ is 25, which enables very high quality decryptions to be obtained from ciphertexts of 500 letters or more. The corresponding test statistic of 0.0254 refutes $H_0$.

### 6.3.6 Final Fitness Function

The results of the previous experiments indicate that the following fitness function is ideal for cryptanalysing substitution ciphers:

$$Cost(k) = 10 \cdot bi(k) + 40 \cdot tri(k) - 25 \cdot dict(k)$$

[10]The $p$-values for $\delta \geq 60$ indicate that these results are significantly worse than those obtained when $\delta = 0$.

The large weight assigned to the trigram measure (relative to the unigram and bigram weights) coincides with the weight values adopted by Clark and Dawson (1997). In addition, the dictionary heuristic has been shown to enhance the quality of decryptions.

The emphasis placed on the trigram measure is in stark contrast with many of the fitness functions proposed in earlier research (as reported in Subsection 2.3.1), which tend to neglect trigram statistics in favour of unigram and bigram statistics. However, the utility of trigrams for successful cryptanalysis has been systematically proven by this experiment.

## 6.4 Evaluating the Cryptanalytic Abilities of the Tool

The suitability of the tool for cryptanalysis is evaluated by performing attacks on a selection of classical ciphers, using the fitness function specified in Subsection 6.3.6.

In the following experiments, the source texts identified in Subsection 6.3.1 are reused as plaintext material and encrypted for each cipher investigated. The correctness of a decrypted text is measured as the percentage of letters in the decrypt which match those of the plaintext.

### 6.4.1 Attacks on Polyalphabetic Substitution Ciphers

Two kinds of polyalphabetic substitution cipher are investigated:

- The first is the Vigenère cipher, which requires a $d$-letter keyword in order to select from a set of Caesar-style shift mappings (Subsection 1.2.1).
- The second (and more general) polyalphabetic cipher is a concatenation of $n$ distinct alphabet permutations. These "mixed alphabets" are applied in sequence to encrypt each letter in the plaintext (in the manner of simple substitution).

  The tool models this form of cipher as a combination of (separate) simple substitution functions; hence, cryptanalysis is performed by searching over the keyspaces associated with each function in parallel.

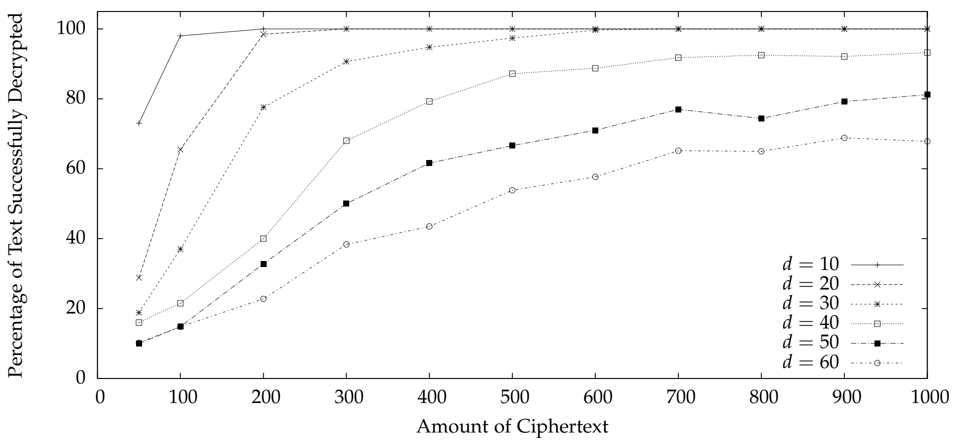


Figure 6.5: A profile of the correctness of decrypts for different Vigenère keyword lengths ($d$).

The results in Figure 6.5 indicate that, for the Vigenère cipher, near-perfect decryptions (> 90% correct) are typically obtained when the amount of available ciphertext exceeds the keyword length by a factor between 10 and 20. This is unsurprising, since, if each position in the keyword is considered individually, only a few candidate letters will correspond to decryptions which possess natural language statistics.

The tool is clearly capable of solving this kind of cipher and, by extrapolation, similar successes would be expected for cryptanalysing related keyword-based polyalphabetic ciphers, such as the Beaufort and Playfair ciphers.

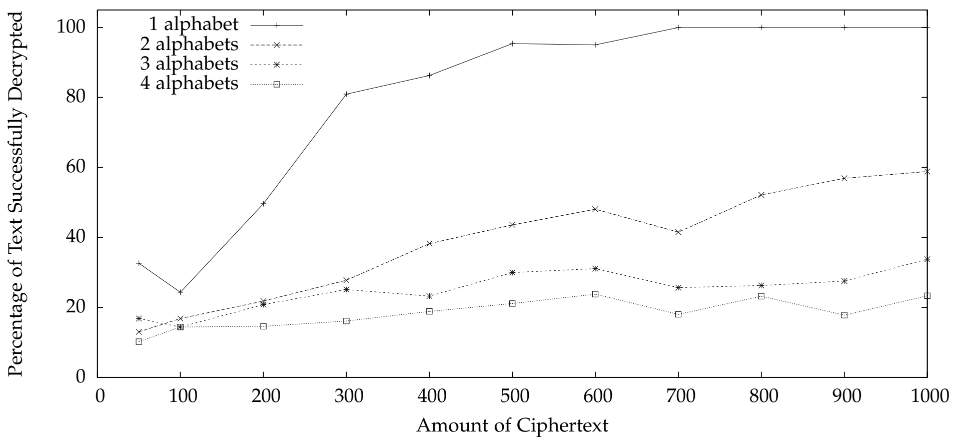


Figure 6.6: A profile of the correctness of decryptions obtained for multiple mixed alphabets.

Cryptanalysis of the mixed alphabet polyalphabetic substitution cipher is moderately successful if the cipher is limited to two substitution alphabets (Figure 6.6). However, when three or more substitution alphabets are present, only a small fraction of the sample ciphertext is decrypted correctly. (Increasing the number of substitution alphabets will widen the keyspace of the cipher and create many more local optima in the search space. Hence, the probability that search will find the correct decryption is diminished.)

The quality of decryptions is not greatly improved for larger amounts of ciphertext, which suggests that the approach taken to cryptanalyse this cipher is the limiting factor. The strategies developed by Jakobsen (1995) and Clark and Dawson (1997) appear to yield better performance than achieved by the tool and therefore warrant further investigation.

### 6.4.2 Attacks on Block Transposition Ciphers

Cryptanalysis of a block transposition cipher (Subsection 1.2.2) was attempted with varying sizes of key permutations. The results obtained are shown in Figure 6.7.

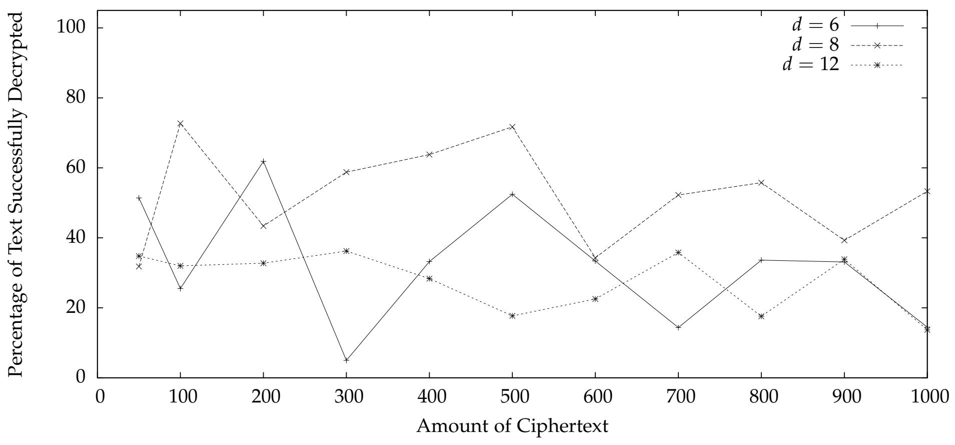


Figure 6.7: A profile of the correctness of decryptions for different sizes of key permutation.

For all but the shortest key sizes, the quality of the decryptions achieved are disappointing. In particular, the average correctness of decryptions fluctuates wildly and shows no improvement for longer ciphertexts. Closer inspection of the results of individual runs indicates that each decryption obtained is either perfect (or almost perfect) or completely incorrect, with a tendency towards the latter.

This "all or nothing" categorisation of the decrypts suggests the outcome of each cryptanalysis run is highly dependent on external factors. The author speculates that the PMX crossover operator (Subsection 4.3.2) introduces too much disruption into the permutation keys, meaning that search is failing to preserve good decryptions in the population.

To eliminate the effect of crossover, the experiment was repeated with simulated annealing as the search algorithm, using the control parameters specified in Table 6.6.

| | | | |
|---|---|---|---|
| Initial Temperature ($T_0$): | 100 | Temperature Iterations ($k$): | 100 |
| Trial Moves (at each $T_k$): | 50 | Geometric Cooling Rate ($\alpha$): | 0.95 |

Table 6.6: Control parameters selected for the simulated annealing algorithm.

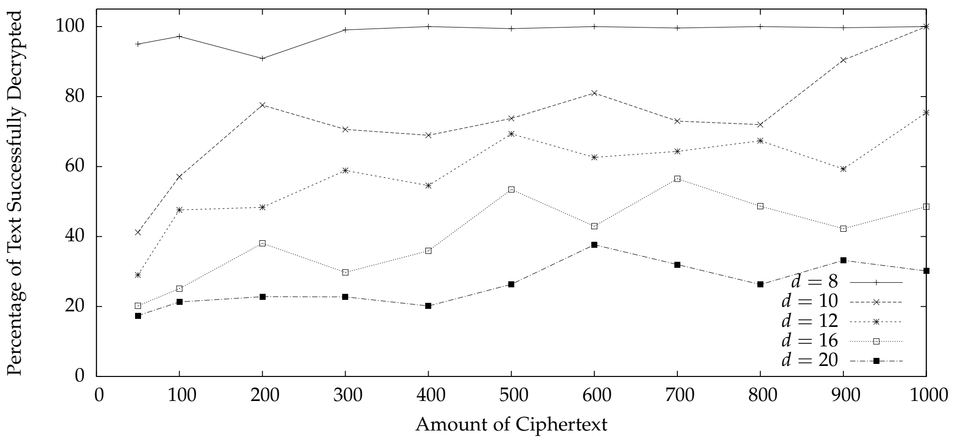


Figure 6.8: A profile of the correctness of decryptions for different sizes of key permutation, with simulated annealing used as the search algorithm.

The results achieved with simulated annealing (Figure 6.8) are much more consistent and exhibit a gradual improvement in decryption correctness for longer ciphertexts. However, the performance for long key permutations ($d \geq 16$) falls somewhat short of the results obtained by Matthews (1993) and Giddy and Safavi-Naini (1994).

It appears that the fitness function is incapable of discriminating between two related transposition keys and thus cannot provide sufficient guidance to the search until a near-correct decryption is reached. The weights of the fitness function were determined by experimentation with a substitution cipher and, hence, there can be no guarantee of their optimality in the context of cryptanalysing transposition ciphers.

## 6.5 Cryptanalysis of Unsolved Cryptograms

In this section, attacks on the unsolved cryptograms described in Section 1.4 are attempted, to determine the behaviour of the cryptanalysis tool on ciphertexts with no apparent solution.

The means of encipherment of these cryptograms is not known, so cryptanalysis can only proceed on the assumption of a particular cipher. Since there are a practically limitless number of methods by which these texts may have been encrypted, only a small fraction of these possible ciphers can be investigated.

### 6.5.1 The Dorabella Cipher

The 87 glyph symbols from the Dorabella cipher (Figure 1.5) were transcribed, by hand, to the following source ciphertext, which was loaded into the cryptanalysis tool:

```
PURYGJEYBPZMDHMSMMJUUNHFPFAZV
EMEJFYBKYRMMRJADEMZMJNVDHFBBFZV
RVJNVRHNWAVJMJNDVAYUAVJYAGY
```

Proceeding on the supposition that Elgar encrypted the message by a combination of simple substitution and block transposition, the cryptanalysis tool produced numerous decryptions. Unfortunately, the decrypted texts are gibberish, although some contain recognisable words:

```
DGNOFTCOLDMEUSEVEETGGHSIDIRMA       UCROWNDOMUBELIEVEENCCGISUSTBA
CECTIOLYONEENTRUCEMETHAUSILLIMA   DEDNSOMPOREERNTLDEBENGALISMMSBA
NATHANSHPRATETHUAROGRATORFO        RANGARIGHTANENGLATOCTANOTWO
```

One should not read too much into the words that appear in these decrypts, as they can be entirely attributed to the optimisation algorithm over-fitting the characteristics of the fitness function (particularly the dictionary heuristic).

It should be noted that 87 letters is barely sufficient for cryptanalysis (Subsection 1.3.1), so the lack of success in decrypting this cipher is unsurprising.

### 6.5.2 The Voynich Manuscript

The text from the first ten folios of the manuscript[11] (totalling approximately 10,000 letters) was subjected to cryptanalysis. This ciphertext was treated as having been encrypted by a simple substitution cipher, although this is unlikely to be the case in reality.

For most runs, the search rapidly converged, resulting in decryptions similar to the following first paragraph:

```
GAORIP.IMAN.AC.ATALLY.PREN.PRECI.OTRCSP.I.MEC.PRENDI
PECI.OMRAC.EC.I.MALC.ORTALLY.PRAC.ACS.OTRAC.OTRAC.DAY
PIALLC.PRSMI.EC.IMALLY.PRED.OTREACI.OTRSP.DACALLY.PA
EELLY.ETSSI.ETSEP.CENETI.OTR*AC.DALLY.ETALLY.EC.EMAY
DALC.I.ORSAC.OTRALLY.OFRAC.OGRALLY
```

This decryption exhibits a high fitness score – it is a good approximation to the expected *n*-gram frequencies – but none of the text is intelligible in any language known to the author (or his supervisor!). Whether the Voynich manuscript is authentic or a hoax, it is not apparent how any meaningful information can be extracted from this text.

### 6.5.3 The 340-Symbol Zodiac Cryptogram

The solved 408-symbol Zodiac cryptogram was encrypted using a homophonic substitution cipher, which suggests that the unsolved 340-symbol cryptogram is similarly encrypted.

Unfortunately, the tool failed to extract any meaningful decryptions from either of the 408-symbol or the 340-symbol ciphertexts. There appears to be a problem in the implementation of the *Relation* key structure (Subsection 5.3.3) with regards to handling homophone symbols. Due to time constraints, the cause of this problem could not be investigated or resolved.

[11] Takeshi Takahashi's transcription of the Voynich manuscript is available at http://www.voynich.com/pages/PagesH.txt

# 7 Evaluation and Further Work

*This chapter evaluates the completed cryptanalysis tool against the objectives stated in Chapter 3. The strengths and weaknesses of the tool are identified and analysed, together with factors that impacted the tool's development. Finally, some potential topics for future research are proposed.*

## 7.1 Appraisal of Project

The project deliverables (Section 3.2) are fulfilled by the completed tool.

### 7.1.1 Analysis of Cipher Model

The cipher model was an unequivocal success. The representation of substitution and transposition operations as functions on text objects worked particularly well in practice, as each individual function could be implemented and tested independently.

The model facilitates the construction of arbitrary product ciphers by coupling text functions together. Although this feature was not utilised in the experiments reported in Chapter 6, it would be useful for modelling multi-step ciphers (such as rotor machines) in future work.

Part of the cipher layer was over-engineered: the support for customisable alphabets in the implementation of text services (Subsection 5.3.1) was unneeded[1]. Furthermore, the representation of each individual text element as a separate object has no discernible advantage over using a numeric representation of symbols and letters in text and, if the cipher layer was to be re-implemented, this latter approach would be favoured for efficiency.

### 7.1.2 Cryptanalysis Capabilities of the Tool

In Chapter 6, a variety of ciphers were subjected to cryptanalysis by the tool. In many cases, the tool produced high quality decrypts, albeit with some exceptions:

- The tool is capable of recovering a high proportion of correct text from a simple substitution cipher with as few as 250 ciphertext letters (Figure 6.4). These results constitute a marked improvement on earlier work: indeed, they are clearly superior to the results reported by Forsyth and Safavi-Naini (1993) and Spillman et al. (1993).
- Good decryptions were achieved in attacks on ciphertexts encrypted by block transposition and the Vigenère cipher, for reasonable key sizes.

  Unfortunately, the tool struggled to perform effective cryptanalysis on block transposition ciphers with large permutation keys and mixed alphabet polyalphabetic substitution ciphers, when compared to the attacks reported by Matthews (1993) and Clark (1998) respectively. In defence of the tool, it should be noted that these researchers specialised their attacks to these particular kinds of ciphers.
- The lack of success of the attempted cryptanalysis of the unsolved cryptograms should not be seen to diminish the tool's capabilities. Indeed, these cryptograms have resisted all previous attacks and, as stated in Section 1.4, they may not be solvable by any means.

[1] In order to perform attacks on the unsolved cryptograms described in Section 1.4, machine-readable versions of these ciphertexts were sourced.

The effectiveness of the tool in attacks on substitution ciphers was greatly improved by incorporating multiple evaluation measures (with appropriate weights) into the fitness function given in Subsection 6.3.6. The author believes that, for the attacks on transposition ciphers, better performance could be achieved by using a different set of weights for the evaluation measures in the fitness function.

There is likely to be further potential for improving the performance of cryptanalysis by tuning the control parameters of the optimisation algorithm. Other possible approaches for attaining better decryptions – in particular, revising the search operators and fitness function – are discussed in Section 7.3.

The tool could be extended to perform attacks on complex polyalphabetic ciphers such as the Enigma machine (and related rotor-based cipher systems). However, due to the additional complexity of these ciphers, their cryptanalysis would necessitate the development of new heuristics for evaluating the correctness of candidate decryptions. (Some research in this area has been performed by Bagnall et al. (1997).)

### 7.1.3 User Interface

The front-end user interface exposes the functionality of the cipher and cryptanalysis layers of the tool (Subsection 5.3.6). Owing to time constraints, further development of the user interface was curtailed in favour of developing the cryptanalysis layer.

Consequentially, there is a large scope for improvement of the user interface, in terms of specific features (such as visualisation of the progress of a cryptanalysis run) and its general usability. Extending the user interface in this way could transform the tool into a program for teaching the principles of classical cryptography and metaheuristic search techniques.

## 7.2 Software Engineering Concerns

The extreme programming (XP) method was selected as the software development model in Section 3.3. In practice, extreme programming worked well for the tool's construction, as it imposed a workable structure on the development process, whilst minimising the documentation overhead[2]. In particular, the adherence to unit testing proved vital for isolating (and resolving) many bugs in the implementation of the cipher and cryptanalysis layers.

As discussed in Section 5.2, several software development tools were adopted to manage various aspects of the development process. The features of the Eclipse IDE – especially the facilities for debugging and code re-factoring – were used heavily during the tool development. These facilities saved a substantial amount of time and effort and significantly improved the code quality of the finished tool.

In retrospect, Java served well as the implementation language for the tool, notwithstanding some concerns regarding execution performance discussed in Subsection 7.3.7. The use of Java enabled the cryptanalysis layer to be interfaced with the ECJ toolkit in a straightforward manner, although some unorthodox coding practices were employed to integrate the implemented key structures (Subsection 5.3.3) with ECJ's string-based representation of the chromosomes of solutions within a population.

[2]The design decisions made for the tool were documented in note form during the construction phase of the project. Chapters 4 and 5 were partially written by expanding these notes.

## 7.3 Potential Topics for Future Investigation

This section describes topics relevant to the project that, for a number of reasons, were not investigated. These include proposals for useful enhancements to the tool, along with ideas and proposals for research that fall outside the scope of the project objectives.

### 7.3.1 Automatic Recognition of Ciphers from Ciphertexts

In order to carry out cryptanalysis on a ciphertext, the tool requires the specification of the cipher used to encrypt the text. Throughout this project, knowledge of the means of encryption of ciphertexts has been assumed (for simplicity). However, this need not be the case, especially when dealing with the cryptograms considered in Section 6.5.

It would be possible to integrate a suite of specialised heuristics into the tool, to recognise the characteristics of particular ciphers:

- There exists a simple test – described in Subsection 1.2.2 – to identify whether a given ciphertext has been produced by a substitution or transposition cipher. In the former case, the tool could then proceed to distinguish between simple (monoalphabetic) and polyalphabetic substitution by analysing the "flatness" of the *n*-gram statistics of the ciphertext. Furthermore, methods such as the Kasiski examination (Subsection 1.3.2) could be used to determine the period of a polyalphabetic cipher automatically.
- Other properties of a ciphertext may be used to determine the encryption method. For example, the ADFGVX cipher (Section A.7) features a ciphertext alphabet consisting of six specific letters. The sole presence of these letters in a ciphertext is therefore indicative of ADFGVX encryption.

It is expected that these heuristics would be adequate for recognising the use of a single cipher transformation in the creation of a given ciphertext. A somewhat more difficult problem – which may require extensions to these heuristics – would be to recognise the multiple substitution and transposition steps that constitute a product cipher.

### 7.3.2 Combining Local and Population-Based Metaheuristics

Many of the attacks reported in Section 6.4 achieved mostly-correct decryptions using a genetic algorithm. One means of enhancing the quality of these decryptions is to increase the population size or the number of generations processed by the GA. An alternative (and computationally more efficient) approach would be to take the best decryption achieved by the GA and perform further refinement using a local search algorithm.

### 7.3.3 Specialising the Search Operators

The mutation and crossover operators implemented in the tool (Subsection 4.3.2) were chosen for their general applicability to a range of key structures. However, there is potential for specialising these search operators to exploit the properties of particular key structures and ciphers, which may enhance the performance of cryptanalysis.

As suggested in Subsection 6.4.2, it may be possible to improve the quality of decryptions for transposition ciphers by using a different crossover operator. One potential candidate is *order crossover*, which preserves the relative positions of elements within a permutation key (Bäck et al., 1997). Alternatively, it would be possible to bias the PMX operator to select crossover points that are closer together, thereby reducing the disruption to the child sequences.

### 7.3.4 Seeding Decrypts with Crib Words

A potential technique for automated cryptanalysis – which, to the author's knowledge, has not been previously considered in the literature – is to seed the population of decrypted texts with candidate decrypts containing words that are suspected to exist in the plaintext. This is practically a re-interpretation of the cribbing technique (Subsection 1.3.3) in the context of search-based cryptanalysis.

With this technique integrated into the tool, the possibilities for attacks on the Dorabella cipher and Zodiac cryptograms are intriguing. Based on the circumstances of each cryptogram, a list of suspected crib words can be drawn up[3]. Before cryptanalysis of a ciphertext commences, a set of candidate keys would be calculated that result in the occurrence of crib words at various positions in the corresponding decrypted text.

### 7.3.5 Refining the Fitness Heuristics

Since the dictionary heuristic does not play any role in the evaluation of decrypts in the early stages of cryptanalysis (Subsection 4.3.1), the tool's run-time efficiency would be improved by introducing this measure only when there is a high probability of finding recognisable words in decryptions. This may be after a certain number of iterations of the optimisation algorithm have elapsed[4], or when a candidate decryption is found that achieves a particularly high score for the $n$-gram measures.

A more radical approach would be to continually adjust the weights of the fitness function during cryptanalysis. For example, only the unigram and bigram measures may be active at the start of the search, but as the search progresses, the fitness function would gradually transition to trigram and dictionary measures. This would encourage the development of decryptions that satisfy the fitness measures as they are introduced – rather than attempting to satisfy all at once – with a view to improving the quality of the final decryption.

For population-based metaheuristics, a potential difficulty with this technique is that, after an adjustment to a weight is made, the fitness score of each candidate decryption key in the population would need to be re-evaluated.

### 7.3.6 Investigating New Metaheuristic Techniques

Most attacks on classical ciphers – including those attempted in Section 6.4 – have been carried out using only simulated annealing and genetic algorithms.

In the years following the publication of these algorithms, several new metaheuristic techniques have been introduced. Their inventors claim that – in many cases – they deliver better performance than earlier algorithms[5], in terms of accuracy and efficiency. Would these new metaheuristics enhance the performance of automated cryptanalysis?

As the tool is integrated with ECJ, the tool has access to the evolutionary metaheuristics supported by ECJ. Hence, the foundations required to facilitate an investigation into the merits of different evolutionary metaheuristics are already present in the tool.

[3]For example, Elgar may have included oblique references to contemporary composers, football or Dora Penny's singing activities (Subsection 1.4.1), whilst the Zodiac cryptogram would be expected to include words such as "kill" or "murder". However, caution must be exercised, as spelling mistakes are sprinkled liberally throughout the Zodiac's letters (Subsection 1.4.3).

[4]This is the approach taken by Clark and Dawson (1997), who adopts a fitness function that includes the trigram measure only for the latter half of a cryptanalysis run.

[5]The work published by Russell et al. (2003) demonstrates the ant colony optimisation (ACO) metaheuristic search algorithm to be capable of breaking columnar transposition ciphers *"which are significantly shorter (up to a factor of about a half) than those tackled by previous metaheuristic methods"*.

### 7.3.7 Improving the Performance of the Cryptanalysis Engine

Unfortunately, the execution time of a cryptanalysis run is far longer than originally anticipated: on a modern dual-core 2GHz processor, a single cryptanalysis run takes approximately 20 seconds to complete. This is consistent with the execution times reported by Forsyth and Safavi-Naini (1993) and Spillman et al. (1993), who had only a fraction of these computing resources at their disposal[6].

One means of improving the run-time speed of cryptanalysis would be to re-implement the cipher and cryptanalysis layers in a language that is compilable to native machine code; however, doing so would be costly in terms of time and effort.

A more pragmatic approach for enhancing the tool's performance would be to introduce parallelism to the cryptanalysis engine. This could be accomplished by adapting the tool to utilise the distributed processing capabilities already present in the ECJ toolkit.

For example, multiple independent cryptanalysis runs could be executed simultaneously and, upon their completion, the highest-fitness decryptions from each of the runs would be collected and presented to the user. It would be possible to extend this model by introducing an element of interbreeding between populations during cryptanalysis, by exchanging the fittest decrypts present in each population at regular intervals.

[6]However, this performance deficit is somewhat offset by the increased processing requirements of the sophisticated heuristics implemented within the tool.

# 8 Conclusions

## 8.1 Summary of Work

The key points arising from this project are as follows:

- A generalised model of classical ciphers has been devised. Multiple forms of substitution and transposition ciphers have been implemented within this model.
- A software tool has been developed to perform automated cryptanalysis of classical ciphers using optimisation techniques guided by a selection of suitable heuristics.
- The effectiveness of the tool has been assessed by carrying out successful attacks on a selection of classical ciphers. For some of these ciphers, the quality of the decrypts produced by the tool exceed those attained by previous reported research. However, there is clear potential for improvement in the approaches used to cryptanalyse other kinds of classical ciphers.

## 8.2 Closing Remarks

This project has culminated in the first software system which is explicitly designed to perform cryptanalysis on a generalised model of classical ciphers.

The work accomplished in this project provides a solid foundation for further research into the application of search-based methods to the cryptanalysis of classical ciphers.

Possible extensions to the tool would address more complex forms of ciphers (featuring combinations of substitution and transposition operations) such as rotor machines and certain types of block ciphers. On an unrelated note, the tool has potential as a teaching aid to illustrate the basic principles of cryptology.

The fundamental characteristic of classical ciphers upon which optimisation-based attacks rely – namely, that related cipher keys yield slightly different decryptions – does not apply for modern block ciphers such as DES and AES.

Modern cryptographic systems are explicitly designed to be extremely difficult to break using any known form of cryptanalysis. It appears that the potential for successful attacks on these ciphers, using only the approaches detailed in this report, is limited. Future work in this area is left as an exercise to the interested reader.

# A A Brief History of Classical Cryptography

*The reader is referred to Kahn (1997), which is the definitive account of the historical development of cryptography. Much of the material in this section is sourced from Kahn (1997).*

## A.1 The Ancient World

The earliest known application of cryptographic principles dates back to ancient Egypt. The tomb of Khnumhotep II[1] bears an inscription that contains non-standard hieroglyphs as phonetic substitutions; the scribe's intention was to convey a sense of the subject's prestige and authority. This technique of message transformation soon proliferated throughout Egypt, with inscriptions obscured as riddles, designed to tempt visitors into reading aloud the blessings contained therein.

Transformation of messages – combined with an element of secrecy – represents a proto-cryptology, in which textual elements are distorted, so that only those with special knowledge can comprehend them. This features in religious traditions: in the Book of Jeremiah, the Atbash cipher[2] is used to encode "`babel`" (the Hebrew form for Babylon) as "`SHESHACH`" (Jeremiah 25:26, 51:41) and "`kashdim`" (the inhabitants of Babylon) is substituted by "`LEB KAMAI`" (Jeremiah 51:1). Likewise, it is suggested that 666[3] – the "Number of the Beast" (Revelation 13:18) – is Hebrew *gematria* for the Roman emperor Nero, the persecutor of early Christians. This would have given the texts plausible deniability, should they have fallen into the hands of the Roman authorities.

The Greek historian Herodotus[4] describes the use of steganography in the Greco-Persian wars. Demaratus, a Greek exile living in Persia, witnessed the assembly of an invasion force to attack Greece. He sent a messenger to his homeland, advising the Greeks of the Persian intentions. To avoid the danger of the Persians intercepting the message, Demaratus wrote it on wooden tablets before covering them with wax, so that they would appear blank. The message reached its destination safely and was revealed to the Greeks, who subsequently mobilised, prepared an ambush and defeated the Persian invaders.

Steganography precipitated the first system of military cryptography. It is said that circa 500BCE, the Greek Spartans introduced the *scytale*, a wooden staff around which a strip of parchment is wrapped. A message is written on the parchment down the length of the staff. The parchment is then unwound: the disconnected letters of the message are meaningless until the parchment is re-wrapped around a scytale of equal diameter, whereby the message is restored (Singh, 2000). The scytale is equivalent to a matrix transposition cipher, with the dimensions of the staff corresponding to the key.

The famous "Caesar cipher" is a monoalphabetic substitution cipher, in which each plaintext character is replaced by the letter a fixed number of places down the alphabet. According to Suetonius, Julius Caesar used it to protect military and political communications with a left shift of 3: for example, the phrase "`veni, vidi, vici`" encrypts to "`SBKF, SFAF,`

[1] Khnumhotep II, a nobleman and provincial governor of the town of Menat-Khufu, circa 1900BCE.

[2] Atbash is a simple substitution cipher: the first letter of the alphabet (in Hebrew, *aleph*) replaces the last (*tav*), the second (*beth*) replaces the penultimate (*shin*), and so on.

[3] Some of the oldest manuscripts have the number 616 instead.

[4] Herodotus (ca. 484BCE – ca. 425 BCE); acclaimed as the "Father of History", but also as the "Father of Lies" by his detractors, for many of his accounts contradict known historical facts in places.

SFZF". Since the number of possible shifts is equal to the number of letters in the alphabet, the keyspace size of the Caesar cipher is 25 (excluding the identity shift), making it trivial to cryptanalyse by enumerating all possibilities.[5]

## A.2 Early Advances

The collapse of the Roman Empire in 476CE and the destruction of the Library of Alexandria plunged Europe into the Dark Ages. However, this period coincided with the Golden Age of Islam, where literacy flourished in an organised and affluent society. This mandated and facilitated the advance of cryptographic techniques, which were used extensively by the Islamic state to protect administrative records (Singh, 2000). The Arab philosopher and scientist Al-Kindī is credited with pioneering[6] cryptanalysis techniques such as probable words and vowel-consonant combinations, cipher classifications and frequency analysis (Al-Kadi, 1992).

During this time, the study of cryptography in the West was limited to the monasteries, where monks attempted to extract messages hidden in biblical texts (Singh, 2000). Slowly, cipher systems were invented independently or introduced from the Arab world. Circa 1250, the Franciscan friar and polymath Roger Bacon published the first text on cryptography [7] in Europe. In particular, this work included an anagrammed recipe for manufacturing gunpowder – an early example of the use of cryptography by alchemists and scientists to keep their discoveries secret (Singh, 2000).

The advent of diplomatic relations between European states in the 15th century brought about the flow of information between countries and their embassies abroad. These communications soon became subject to espionage: then, as now, no government had qualms about intercepting and reading the secrets of other states (Belfield, 2006). By the end of the century, most countries in Europe employed full-time schools of cipher secretaries to protect their messages and decrypt foreign dispatches.

## A.3 Polyalphabetic Ciphers

In the 1460s, without a cipher school of its own, the Vatican approached the great Renaissance architect and scholar Leon Battista Alberti to devise methods for reading the communications of its enemies. Already in his sixties, Alberti produced a treatise on cryptanalytic techniques, then proceeded to invent the polyalphabetic substitution cipher in the form of the cipher disc.

The cipher disc is a device made of two circular plates, one smaller and mounted concentrically above the larger. The circumference of each plate is divided into equally-sized sectors for each letter of the alphabet (in any order), representing a substitution mapping between plaintext and ciphertext. By turning one plate relative to the other, a different cipher mapping is produced, giving rise to the first polyalphabetic cipher system. Another innovation was the inclusion of code numbers on the disc, facilitating their encryption along with the text.

*Polygraphiæ* (1508), written by the abbot and occultist Johannes Trithemius, describes a polyalphabetic cipher based on the *tabula recta* (Figure 1.2), a square table containing all shifts

[5]It appears that Bernardo Provenzano, the reputed "boss of bosses" of the Sicilian Mafia, was unaware of this shortcoming. Provenzano used a variant of the Caesar cipher for written orders to his associates: this enabled Italian investigators to decrypt these messages, contributing to his arrest in April 2006. Fittingly, a contemporary of Provenzano is reported to have once remarked that: *"He shoots like a god, shame he has the brains of a chicken..."*

[6]A copy of Al-Kindī's text, *A Manuscript on Deciphering Cryptographic Messages*, was rediscovered in 1987, in the Sulaimaniyah Ottoman Archive in Istanbul (Al-Kadi, 1992).

[7]This is the *Epistle on the Secret Works of Art and the Nullity of Magic*, which describes seven (simple) ciphers. Bacon argues for the right of everyone to secure communication: *"The man is insane who writes a secret in any other way than one which will conceal it from the vulgar and make it intelligible only with difficulty even to scientific men and earnest students."*

of the Latin alphabet, in the manner of the Caesar cipher. To encrypt a message, the first letter is encrypted using the first row of the table, the second letter with the second row, and so on: for example, the phrase "`black magic`" encrypts to "`BMCFO RGNQL`". A significant advancement over Trithemius' use of the *tabula recta*, published by Giovan Battista Bellaso in 1553, is now known[8] as the Vigenère cipher (see Subsection 1.2.1).

Although the ideas of Alberti, Trithemius and Bellaso far surpassed the contemporary cryptographic systems in sophistication, they were shunned by the European cipher secretaries, because polyalphabetic substitution is a slow process to perform by hand, whilst a single mistake in application will garble the ciphertext, making its decryption impossible.

## A.4 The Babington Plot

The reign of Queen Elizabeth I of England was a time of deep religious tension. Elizabeth, a Protestant, was resented by her Catholic subjects, who considered her cousin Mary Stuart[9] to be the rightful Queen of England. In 1567, Mary was forced to abdicate the Scottish throne and flee to England, only to be placed under house arrest for the following eighteen years. During this time, Mary was implicated in several plots against Elizabeth's rule; however, Elizabeth refused to sanction her execution, not least because she was her relative and a fellow monarch.

A young priest, Gilbert Gifford, was recruited to establish a secret communications channel between Mary and the French ambassador. Mary took elaborate measures to ensure the security of her correspondence, including the use of a nomenclator to encrypt her messages.

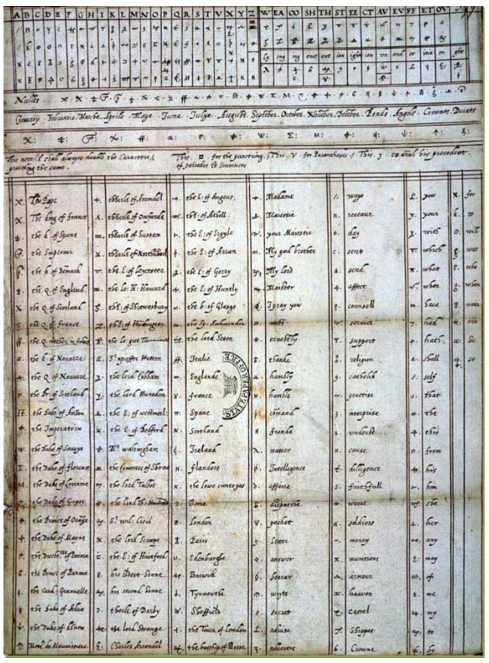


Figure A.1: Mary's nomenclator cipher. (Source: the UK National Archives website.)

In March 1586, Anthony Babington, a Catholic nobleman who held particular grievances toward the Protestant state, gathered six confidants and hatched an ambitious plan to assassinate Elizabeth and install Mary on the throne. Although the Vatican had sanctioned the overthrow

[8]When the cipher gained prominence in the 19th century, it was misattributed to Blaise de Vigenère who, in his *Traicté des Chiffres* (1586), had instead proposed a refinement of the method – the *autokey cipher* – in which the plaintext is *itself* be used as the key.

[9]Mary I of Scotland (1542 – 1587); popularly known as Mary, Queen of Scots.

of Elizabeth[10], the plotters resolved to seek Mary's authorisation before taking action. Early in July, Babington exchanged several letters with Mary specifying the details of the plan.

What neither Mary nor Babington knew was that Gilbert Gifford was a double agent who was diverting their correspondence to his controller Sir Francis Walsingham, the principal secretary to Queen Elizabeth and ruthless state spymaster. Walsingham assigned his star cryptanalyst Thomas Phelippes to break Mary's nomenclator and decipher the messages, but waited for Mary to incriminate herself before taking action. Having received the letter from Mary in which she acquiesced to the plot, Walsingham had Phelippes forge an encrypted postscript before it was relayed to Babington, asking for *"the names and qualities of the six gentlemen which are to accomplish the designment"*.

Babington and the other conspirators were rounded up and, on 18th September 1586, were convicted of high treason and sentenced to death. Later, Mary was tried on the same charge and, on the evidence of her deciphered letters, found guilty. Mary's part in the conspiracy convinced Queen Elizabeth that she would remain a threat as long as she remained alive. Her death warrant was signed in February 1587.

## A.5 The Black Chambers

The cryptographic career of Antoine Rossignol began in April 1628, when Henri II de Bourbon, the Prince of Condé, laid siege to the Huguenot stronghold town of Réalmont. Their defences unbreached, the town's inhabitants put up fierce resistance, but were in desperate need of munitions. Unable to hold out for much longer without outside support, the Huguenot commander sent an enciphered message to summon reinforcements, but its courier was caught when attempting to sneak through the blockade. Since Condé's officers were unable to decrypt the message, Rossignol – who was reputed to have an interest in ciphers – was summoned to examine the message and, before nightfall, he had extracted its contents. Having now learnt that Réalmont was on the brink of capitulation, Condé had the decrypted message returned to the defenders, on the understanding that the siege would not be lifted. Knowing that defeat was now inevitable, the Huguenots promptly surrendered.

When news of this victory reached Cardinal Richelieu, he immediately appointed Rossignol to his court and set him to work on other intercepted Huguenot messages. This resulted in similar successes, quickly establishing Rossignol as the foremost cryptographer in Europe. His expertise helped to strengthen the French nomenclator ciphers in use at the time, culminating in the design of the *Great Cipher* of Louis XIV, a nomenclator which only Rossignol, his son and grandson fully mastered[11].

Together with other French cryptographers, the Rossignols established the *cabinet noir* (black chamber), a government office through which foreign diplomatic mail was routed: letters were opened and their contents duplicated, before being resealed and returned to the postal system. In the 1700s, many European powers followed France's lead and established their own black chambers. The *Geheime Kabinets-Kanzlei* in Vienna was the most efficient of all: it not only supplied the intelligence which directed Austrian foreign policy, but also sold the information it harvested to friendly nations. (Singh, 2000)

The black chambers represented cryptography's "industrial revolution", with large-scale organised cryptanalytic efforts rendering insecure all primitive ciphers and forcing cipher secretaries to adopt tougher systems such as the Vigenère cipher. In turn, the focus of the black chambers shifted to intercepting and reading personal mail. The political reforms of

[10]*Regnans in Excelsis*, the papal bull issued by Pope Pius V in 1570, demanded that Elizabeth was to be *"...deprived of her pretended title to the crown and of all lordship, dignity and privilege whatsoever."*

[11]The Great Cipher was used to protect the French state archive. It resisted all enemy cryptanalytic efforts and remained unbroken until the 1890s, when it attracted the interest of the French cryptanalyst Étienne Bazeries, who devoted three years of his life to its solution. (Singh, 2000)

the mid-19th century swept away the old authoritarian European order and, in 1844, public pressure forced the British Parliament to dissolve its black chamber. Four years later, the governments of France and Austria also abolished their black chambers.

## A.6 Fair Play in Love and War

As the era of the black chambers was brought to an end, a revolutionary technology heralded a new age of cryptography. This was the electric telegraph, which facilitated instantaneous transmission of messages over long distances. By the 1870s, commercial telegraph networks extended to every continent, making global communication possible for the first time.

Although the telegraph operators were sworn to secrecy, their customers desired a means of keeping the contents of their telegrams private, in the same way that the envelope provides privacy of correspondence in the postal system. This resulted in the development of many codes and ciphers to guard against casual snooping; one of the strongest being the *Playfair cipher*, invented by Charles Wheatstone[12] and popularised by Lyon Playfair, Baron of St. Andrews. This is a digraphic substitution cipher, where letters are enciphered as pairs, spreading the ciphertext characteristics over 625 ($25^2$) bigrams, rather than 25 letters[13].

The heightened public awareness of cryptographic techniques was not limited to their use in telegraph messages; soon, ciphers were being used for a variety of purposes. Since romantic expression was forbidden by Victorian society, young lovers communicated by exchanging coded notes of affection in the "personal columns" of newspapers. Unsurprisingly, it became an amusement of amateur cryptographers to break these messages and, sometimes, to surreptitiously insert a reply of their own[14].

Prior to the introduction of the telegraph, military strategy was impeded by an inability to coordinate forces in battle from a distance. The telegraph saw limited action in the Crimea, but the full extent of its effectiveness was shown in the American Civil War[15]. However, signal lines can be wiretapped, so it became necessary to employ *field ciphers* to ensure secure communication. A field cipher provides a compromise between simplicity and security: it must be usable (to a high degree of accuracy) by soldiers without extensive training, but also be resistant to enemy cryptanalysis until its secret loses all practical value. Early examples of field ciphers include the polyalphabetic cipher disc (Section A.3) which served both Union and Confederate forces in the Civil War; the Playfair cipher, reportedly used by the British in the Boer Wars and the St. Cyr slide rule (Pratt, 1939), a device to speed up the encryption and decryption processes of the Vigenère cipher.

## A.7 The First World War

At the turn of the century, the invention of the radio – the "wireless telegraph" – attracted worldwide interest, for it offered the capability of the telegraph without needing an established line of communication. For the military, however, the very nature of radio was its greatest drawback, because it was impossible to avoid every transmission from being broadcasted to the enemy. More than ever, a secure cipher was required, but none was forthcoming.

[12]Charles Wheatstone (1802 – 1875); an English physicist and inventor who, with William Fothergill Cooke, was a major figure in the development of the British telegraph system, receiving a knighthood for this work in 1868.

[13]Playfair is constructed as a $5 \times 5$ alphabet square, with the 'I' and 'J' coinciding. However, it is still a monoalphabetic cipher, which makes it susceptible to bigram frequency analysis.

[14]Amongst others, Charles Wheatstone and Lyon Playfair took part in this activity: having broken into one such correspondence, they composed a message in its cipher, prompting the response: *"Dear Charlie: write no more. Our cipher is discovered."* (Bauer, 1997)

[15]In his memoirs, General William T. Sherman wrote that *"the value of the magnetic telegraph in war cannot be exaggerated"*.

Cryptography was in disarray. Kasiski had demolished the security of the "unbreakable" Vigenère cipher and related polyalphabetic systems (such as the Beaufort cipher). For the first time since Rossignol, the cryptanalysts had the upper hand and matters did not improve for those who wished to maintain the secrecy of their communications[16]. The increasing tensions between the European nations compelled them to develop larger and more elaborate code systems than ever before (Pratt, 1939).

By 1914, Europe was on the brink of war. The assassination of Archduke Franz Ferdinand of Austria sparked the inevitable chain reaction. The opposing powers had prepared well materially, but their communication systems were staggeringly ineffective. In August 1914, the Russian commanders Rennenkampf and Samsonov, having been issued with different versions of the Russian cipher, foolishly resorted to exchanging the details of their troop movements in unencrypted form. These messages were intercepted by the Germans, who changed their strategy in light of this evidence, culminating in the destruction of the Russian Second Army at the Battle of Tannenberg (Pratt, 1939).

One early success of the Entente powers was the Russian Navy's capture of codebooks from the German cruiser *Magdeburg*. This windfall provided "Room 40", the British Admiralty's newly established decoding department, not only with the current German naval codes, but also with an insight into the system on which these codes were built (Pratt, 1939). For the next two years, the British were able to read their enemy's coded messages without difficulty until, in the aftermath of the naval battle of Jutland, the Germans grew suspicious and changed their system in August 1916 (Pratt, 1939).

Amongst its many successes, Room 40 is credited with the most important decipherment of the 20th century. In January 1917, the British intercepted a coded telegram[17] from the German Foreign Office, addressed to the German embassy in Mexico City. This aroused the suspicions of Room 40, which assigned two cryptanalysts – Nigel de Grey and William Montgomery – to its decipherment. History knows the result as the infamous *Zimmermann telegram*, which detailed a German plan to commence unrestricted submarine warfare in the Atlantic against American interests[18]. When the British handed a copy to the United States ambassador (and the press), the American policy of neutrality towards Germany became untenable, and soon led to the United States declaring war against Germany in April 1917.

The Germans introduced their ADFGX[19] field cipher just prior to the commencement of the Spring Offensive[20]. Initial attempts to break the cipher were unsuccessful, but there was soon an abundance of intercepts reporting progress of the German attack. Georges Painvin, the best cryptanalyst in the French service, worked on the intercepted ciphertexts continuously for two months, to the detriment of his health. He achieved his first complete break in early April, with other decrypts flowing regularly thereafter. These decrypts enabled the French to pinpoint the target of the German bombardment of June, allowing them to sidestep the offensive and launch a counterattack.

[16]In the wake of the untimely death of President Félix Faure in 1899, the resurgent French royalists made plans to stage a coup. However, their communications, encrypted with the Beaufort cipher (a variant of the Vigenère cipher), were promptly broken by Étienne Bazeries using a combination of the Kasiski test and probable words. This gave the authorities forewarning of the uprising, allowing them to act decisively: the key conspirators were arrested and, on the testimony of Bazeries, were exiled in 1900 (Pratt, 1939).

[17]The telegram was encoded in the German code 0075, known to Room 40 since July 1916.

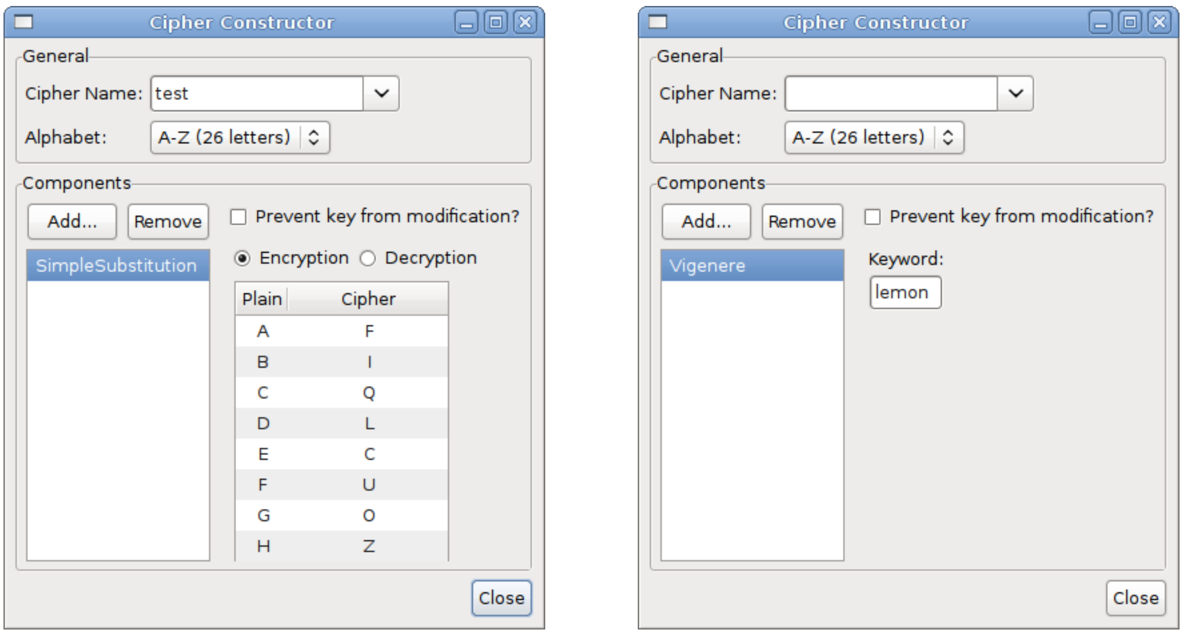
[18]To counter the inevitable retaliation, Germany proposed to ally with Mexico and finance a Mexican invasion of the United States. (Mexico had ceded territory to the United States under the terms of the Treaty of Guadalupe Hidalgo in 1848.)

[19]ADFGX is a fractionating (Polybius square substitution) transposition cipher, named for the five letters that comprise its ciphertext alphabet. It was extended to six letters, becoming ADFGVX.

[20]The Spring Offensive of 1918 represented Germany's final push for victory, before the overwhelming resources of the United States could be deployed. The German forces made the largest territorial gains on the Western Front by either side during the entire war, but suffered huge losses which could not be replenished.

# B Screenshots of Tool User Interface (unmarked)

*An overview of the tool's user interface is given in Subsection 5.3.6.*



Cipher Constructor

General

Cipher Name: test

Alphabet: A-Z (26 letters)

Components

Prevent key from modification? ☐

SimpleSubstitution

[✓ Encryption] Decryption

| Plain | Cipher |
|---|---|
| A | F |
| B | I |
| C | Q |
| D | L |
| E | C |
| F | U |
| G | O |
| H | Z |

Cipher Constructor

General

Cipher Name:

Alphabet: A-Z (26 letters)

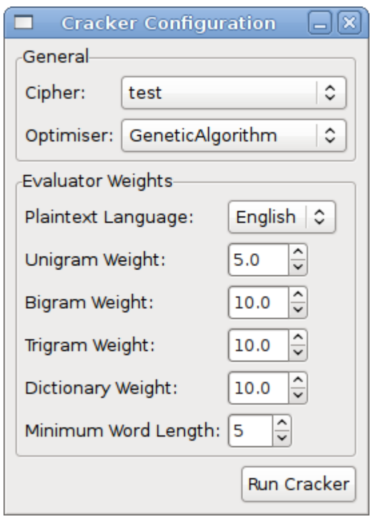
Components

Prevent key from modification? ☐

Vigenere

Keyword: lemon

Figure B.1: A dialog to construct cipher "pipelines", by specifying a list of cipher functions and (optionally) their keys. Part of the key shown in Figure 1.1 is visible.



Cracker Configuration

General

Cipher: test

Optimiser: GeneticAlgorithm

Evaluator Weights

Plaintext Language: English

Unigram Weight: 5.0

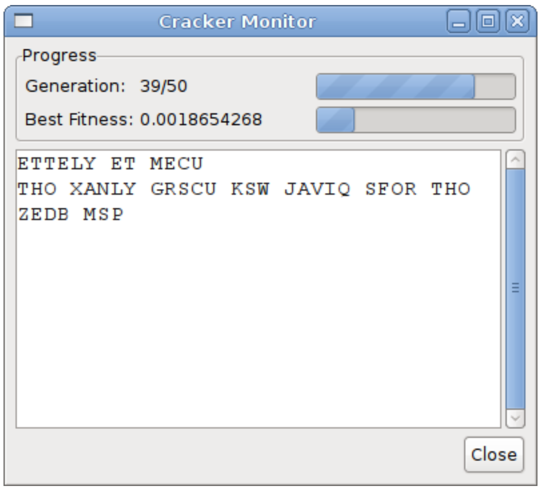
Bigram Weight: 10.0

Trigram Weight: 10.0

Dictionary Weight: 10.0

Minimum Word Length: 5

Figure B.2: A dialog to set weight values for each component of the fitness function.

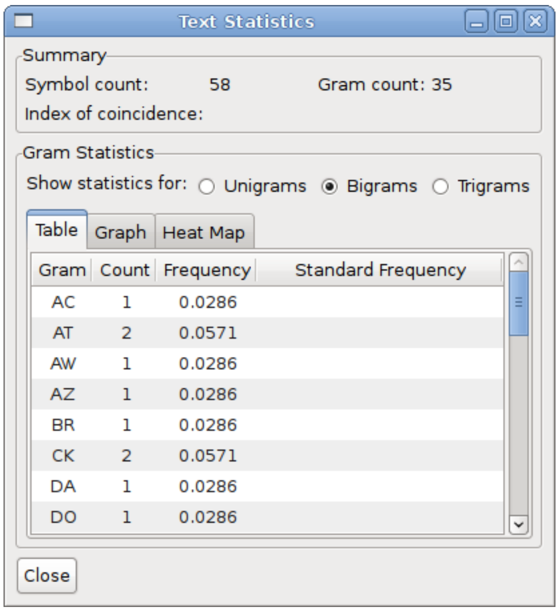


Generation: 39/50

Best Fitness: 0.0018654268

```
ETTELY ET MECU
THO XANLY GRSCU KSW JAVIQ SFOR THO
ZEDB MSP
```

Figure B.3: A dialog to show the progress of a cryptanalysis run.



Symbol count: 58 Gram count: 35

Index of coincidence:

Show statistics for: Unigrams [✓ Bigrams] Trigrams

| Gram | Count | Frequency | Standard Frequency |
|---|---|---|---|
| AC | 1 | 0.0286 | |
| AT | 2 | 0.0571 | |
| AW | 1 | 0.0286 | |
| AZ | 1 | 0.0286 | |
| BR | 1 | 0.0286 | |
| CK | 2 | 0.0571 | |
| DA | 1 | 0.0286 | |
| DO | 1 | 0.0286 | |

Figure B.4: A dialog to show the *n*-gram statistics of a text.

# C Examples of Decrypted Texts (unmarked)

```
ALICEWASBEGINNINGTOGETVERYTIREDOFSITTINGBYHERSISTERONTHEBANKANDOF
HAVINGNOTHINGTODOONCEORTWICESHEHADPEEPEDINTOTHEBOOKHERSISTERWAS
READINGBUTITHADNOPICTURESORCONVERSATIONSINITANDWHATISTHEUSEOFABOOK
THOUGHTALICEWITHOUTPICTURESORCONVERSATIONSOSHEWASCONSIDERINGINHER
OWNMINDASWELLASSHECOULDFORTHEHOTDAYMADEHERFEELVERYSLEEPYANDSTUPID
WHETHERTHEPLEASUREOFMAKINGADAISYCHAINWOULDBEWORTHTHETROUBLEOF
GETTINGUPANDPICKINGTHEDAISIESWHENSUDDENLYAWHITERABBITWITHPINKEYES
RANCLOSEBYHERTHEREWASNOTHINGSOVERYREMARKABLEINTHAT
```

Figure C.1: The correct plaintext: the first 500 letters of *Alice's Adventures in Wonderland*.

```
AYICEWASBEGINNINGTOGETVEMUTIMEDOPSITTINGBUHEMSISTEMONTHEBANFANDOP
HAVINGNOTHINGTODOONCEOMTWICESHEHADREEREDINTOTHEBOOFHEMSISTEMWAS
MEADINGBLTITHADNORICTLMESOMCONVEMSATIONSINITANDWHATISTHELSEOPABOOF
THOLGHTAYICEWITHOLTRICTLMESOMCONVEMSATIONSOSHEWASCONSIDEMINGINHEM
OWNXINDASWEYYASSHECOLYDPOMTHEHOTDAUXADEHEMPEEYVEMUSYEERUANDSTLRID
WHETHEMTHERYEASLMEOPXAFINGADAISUCHAINWOLYDBEWOMTHTHETMOLBYEOP
GETTINGLRANDRICFINGTHEDAISIESWHENSLDDENYUAWHITEMABBITWITHRINFEUES
MANCYOSEBUHEMTHEMEWASNOTHINGSOVEMUMEXAMFABYEINTHAT
```

Figure C.2: A sample decryption with approximately 80% letters correct.

```
AUIMECALFEGINNINGSOGESVERYSIREDOBLISSINGFYTERLILSERONSTEFANKANDOB
TAVINGNOSTINGSODOONMEORSCIMELTETADHEEHEDINSOSTEFOOKTERLILSERCAL
READINGFPSISTADNOHIMSPRELORMONVERLASIONLINISANDCTASILSTEPLEOBAFOOK
STOPGTSAUIMECISTOPSHIMSPRELORMONVERLASIONLOLTECALMONLIDERINGINTER
OCNWINDALCEUUALLTEMOPUDBORSTETOSDAYWADETERBEEUVERYLUEEHYANDLSPHID
CTESTERSTEHUEALPREOBWAKINGADAILYMTAINCOPUDFECORSTSTESROPFUEOB
GESSINGPHANDHIMKINGSTEDAILIELCTENLPDDENUYACTISERAFFISCISTHINKEYEL
RANMUOLEFYTERSTERECALNOSTINGLOVERYREWARKAFUEINSTAS
```

Figure C.3: A sample decryption with approximately 60% letters correct.

# Bibliography

Alfred V. Aho and Margaret J. Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975. ISSN 0001-0782. URL http://dx.doi.org/10.1145/360825.360855.

Ibrahim A. Al-Kadi. The origins of cryptology: The Arab contributions. *Cryptologia*, 16(2): 97–126, 1992. URL http://dx.doi.org/10.1080/0161-119291866801.

Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz. *Handbook of Evolutionary Computation*. CRC Press, 1997. ISBN 0-7503-0895-8.

A. J. Bagnall. The applications of genetic algorithms in cryptanalysis. Master's thesis, School of Information Systems, University of East Anglia, 1996. URL http://citeseer.ist.psu.edu/bagnall96applications.html.

A. J. Bagnall, G. P. McKeown, and V. J. Rayward-Smith. The cryptanalysis of a three rotor machine using a genetic algorithm. In Thomas Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA97)*, San Francisco, CA, 1997. Morgan Kaufmann. URL http://www.sys.uea.ac.uk/~ajb/PDF/gecco97.pdf.

Michael Barlow. The Voynich manuscript – by Voynich? *Cryptologia*, 10(4):210–216, 1986. ISSN 0161-1194. URL http://dx.doi.org/10.1080/0161-118691861010.

Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. The SEI Series in Software Engineering. Addison-Wesley Professional, second edition, 2003. ISBN 0-321-15495-9.

Friedrich Ludwig Bauer. *Decrypted Secrets: Methods and Maxims of Cryptology*. Springer-Verlag New York, Inc., New York, NY, USA, 1997. ISBN 3-540-60418-9.

Kent Beck and Cynthia Andres. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, second edition, 2004. ISBN 0-321-27865-8.

Henry Beker and Fred Piper. *Cipher Systems: The Protection of Communications*. Northwood Books, London, 1982. ISBN 0-7198-2611-X.

Richard Belfield. *Can You Crack The Enigma Code?* Orion Books, 2006. ISBN 0-7528-7526-4.

Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003. URL http://iridia.ulb.ac.be/~meta/newsite/downloads/ACSUR-blum-roli.pdf.

Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley Professional, 20th anniversary edition, 1995. ISBN 0-201-83595-9.

Robert F. Churchhouse. *Codes and Ciphers: Julius Caesar, the Enigma, and the Internet*. Cambridge University Press, 2001. ISBN 0-521-00890-5.

Andrew Clark. *Optimisation Heuristics for Cryptology*. PhD thesis, Queensland University of Technology, 1998. URL http://sky.fit.qut.edu.au/~clarka/papers/thesis-ac.pdf.

Andrew Clark and Ed Dawson. A parallel genetic algorithm for cryptanalysis of the polyalphabetic substitution cipher. *Cryptologia*, 21(2):129–138, 1997.

Andrew Clark, Ed Dawson, and Helen Bergen. Combinatorial optimization and the knapsack cipher. *Cryptologia*, 20(4):85–93, 1996.

John A. Clark. *Metaheuristic Search as a Cryptological Tool*. PhD thesis, University of York, 2002. URL http://www.cs.york.ac.uk/ftpdir/reports/YCST-2002-07.pdf.

John A. Clark. Nature-inspired cryptography: Past, present and future. In *The 2003 Congress on Evolutionary Computation (CEC-2003)*, volume 3, pages 1647–1654. IEEE, December 2003. ISBN 0-7803-7804-0. URL http://www-users.cs.york.ac.uk/~jac/PublishedPapers/CEC-03/invited.pdf.

Stephen Cook. The P versus NP problem. Prepared for the Clay Mathematics Institute for the Millennium Prize Problems., 2000. URL http://www.claymath.org/millennium/P_vs_NP/pvsnp.pdf.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001. ISBN 0-262-53196-8.

Captain Prescott H. Currier. Papers on the Voynich manuscript. 1976. URL http://www.voynich.net/reeds/currier/p.ps.

Richard Dawkins. *The Blind Watchmaker*. Norton & Company, Inc, 1986. ISBN 0-393-31570-3.

Dorothy Elizabeth Robling Denning. *Cryptography and Data Security*. Addison-Wesley Longman Publishing Company, Inc., Boston, MA, USA, 1982. ISBN 0-201-10150-5.

Agoston E. Eiben and James E. Smith. *Introduction to Evolutionary Computing*. Natural Computing Series. Springer, 2003. ISBN 3-540-40184-9.

William S. Forsyth and Reihaneh Safavi-Naini. Automated cryptanalysis of substitution ciphers. *Cryptologia*, 17(4):407–418, 1993. URL http://dx.doi.org/10.1080/0161-119391868033.

Martin Fowler and Jim Highsmith. The Agile Manifesto. *Software Development Magazine*, August 2001. URL http://www.ddj.com/architect/184414755.

Helen Fouché Gaines. *Cryptanalysis: a Study of Ciphers and Their Solutions*. Dover Publications, Inc., 1956. ISBN 0-486-20097-3.

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995. ISBN 0-201-63361-2.

Poonam Garg and Aditya Shastri. An improved cryptanalytic attack on knapsack cipher using genetic algorithm. *International Journal of Information Technology*, 3(3):145–152, 2007. URL http://www.waset.org/ijit/v3/v3-3-21.pdf.

Michel Gendreau and Jean-Yves Potvin. Metaheuristics in combinatorial optimization. *Annals of Operations Research*, 140(1):189–213, 2005. URL http://dx.doi.org/10.1007/s10479-005-3971-7.

J. P. Giddy and Reihaneh Safavi-Naini. Automated cryptanalysis of transposition ciphers. *The Computer Journal*, 37(5):429–436, 1994. URL http://dx.doi.org/10.1093/comjnl/37.5.429.

Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5):533–549, 1986. ISSN 0305-0548. URL http://dx.doi.org/10.1016/0305-0548(86)90048-1.

Andrew Hodges. *Alan Turing: The Enigma*. Vintage, 1992 edition, 1992. ISBN 0-09-911641-3.

Thomas Jakobsen. A fast method for cryptanalysis of substitution ciphers. *Cryptologia*, 19(3): 265–274, 1995. URL http://dx.doi.org/10.1080/0161-119591883944.

Kevin Jones. Breaking Elgar's enigmatic code. *New Scientist*, 2479:56, December 2004. URL http://www.newscientist.com/article.ns?id=mg18424792.600.

David Kahn. *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet.* Simon & Schuster, 1997. ISBN 0-684-83130-9.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. URL http://www.cs.virginia.edu/cs432/documents/sa-1983.pdf.

Joanna Kolodziejczyk. The application of genetic algorithm in cryptoanalysis of knapsack cipher. In *European School on Genetic Algorithms (Eurogen97)*, 1997. URL http://ceani.ulpgc.es/ingenetcd/eurogenxx/eurogen97/contributed/kolodziejczyk/ps/kolodziejczyk.ps.

Louis Kruh. Still waiting to be solved: Elgar's 1897 cipher message. *Cryptologia*, 22(2):97–98, 1998. ISSN 0161-1194. URL http://dx.doi.org/10.1080/0161-119891886803.

Gabriel Landini. Evidence of linguistic structure in the Voynich manuscript using spectral analysis. *Cryptologia*, 25(4):275–295, 2001. ISSN 0161-1194. URL http://dx.doi.org/10.1080/0161-110191889932.

O. Lebedko and A. Topchy. On efficiency of genetic cryptanalysis for knapsack ciphers. In *Poster Proceedings of ACDM 98*, 1998. URL http://www.msu.edu/~topchyal/acdm98.ps.

Sean Luke. ECJ 16: a Java-based evolutionary computation research system, 2007. URL http://www.cs.gmu.edu/~eclab/projects/ecj/.

Carmine Mangione. Performance tests show Java as fast as C++. February 1998. URL http://www.javaworld.com/jw-02-1998/jw-02-jperf.html.

Robert A. J. Matthews. The use of genetic algorithms in cryptanalysis. *Cryptologia*, 17(2): 187–201, 1993. ISSN 0161-1194. URL http://dx.doi.org/10.1080/0161-119391867863.

A. McIntyre, G. C. Wilson, and M. I. Heywood. Resource review: Three open source systems for evolving programs – Lilgp, ECJ and Grammatical Evolution. *Genetic Programming and Evolvable Machines*, 5(1):103–105, March 2004. URL http://dx.doi.org/10.1023/B:GENP.0000017053.10351.dc.

Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, October 1996. ISBN 0-8493-8523-7. URL http://www.cacr.math.uwaterloo.ca/hac/.

Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953. URL http://dx.doi.org/10.1063/1.1699114.

Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, third edition, 1996. ISBN 3-540-60676-9.

Richard A. Mollin. *Codes: The Guide to Secrecy From Ancient to Modern Times.* CRC Press, 2005. ISBN 1-58488-470-3.

Nate Nystrom. Performance of scripting languages, August 2007. URL http://www.cs.purdue.edu/homes/sbarakat/cs456/Scripting.pdf. Lecture slides.

Fletcher Pratt. *Secret and Urgent: the Story of Codes and Ciphers.* Robert Hale, London, 1939.

Gordon Rugg. An elegant hoax? A possible solution to the Voynich manuscript. *Cryptologia*, 28(1):31–46, 2004. ISSN 0161-1194. URL http://dx.doi.org/10.1080/0161-110491892755.

Matthew D. Russell, John A. Clark, and Susan Stepney. Making the most of two heuristics: Breaking transposition ciphers with ants. In *Proceedings of the 2003 Congress on Evolutionary Computation*, volume 4, pages 2653–2658, December 2003. URL http://www-users.cs.york.ac.uk/~jac/PublishedPapers/CEC-03/ant-paper1.pdf.

Eric Sams. Elgar's cipher letter to Dorabella. *The Musical Times*, 111(1524):151–154, 1970. URL http://dx.doi.org/10.2307/956733.

Eric Sams. Elgar's Enigmas. *Music & Letters*, 78(3):410–415, 1997. URL http://dx.doi.org/10.1093/ml/78.3.410.

Matthew Scarpino, Stephen Holder, Stanford Ng, and Laurent Mihalkovic. *SWT/JFace in Action: GUI Design with Eclipse 3.0*. Manning Publications, 2004. ISBN 1-932394-27-3.

Andreas Schinner. The Voynich manuscript: Evidence of the hoax hypothesis. *Cryptologia*, 31 (2):95–107, 2007. ISSN 0161-1194. URL http://dx.doi.org/10.1080/01611190601133539.

Claude Elwood Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28(4):656–715, 1949.

Simon Singh. *The Code Book: The Secret History of Codes and Code-breaking*. Fourth Estate, 2000. ISBN 1-85702-889-9.

Ian Sommerville. *Software Engineering*. Addison Wesley, sixth edition, 2001. ISBN 0-201-39815-X.

Richard Spillman. Cryptanalysis of knapsack ciphers using genetic algorithms. *Cryptologia*, 17 (4):367–377, 1993. ISSN 0161-1194. URL http://dx.doi.org/10.1080/0161-119391867999.

Richard Spillman, Mark Janssen, Bob Nelson, and Martin Kepner. Use of a genetic algorithm in the cryptanalysis of simple substitution ciphers. *Cryptologia*, 17(1):31–44, 1993. ISSN 0161-1194. URL http://dx.doi.org/10.1080/0161-119391867746.

Douglas Robert Stinson. *Cryptography: Theory and Practice*. CRC Press, second edition, 2002. ISBN 1-58488-206-9.

René Zandbergen. Known history of the Voynich manuscript. 2004. URL http://voynich.nu/history.html.