The Applications of Genetic Algorithms in Cryptanalysis

by

A. J. Bagnall

A thesis submitted for the degree of Master of Science

by Research

School of Information Systems

University of East Anglia, 1996

Abstract

This thesis describes a method of deciphering messages encrypted with rotor machines utilising a Genetic Algorithm to search the keyspace. A fitness measure based on the phi test for non randomness of text is described and the results show that an unknown three rotor machine can generally be cryptanalysed with about 4000 letters of ciphertext. The results are compared to those given using a previously published technique and found to be superior.

Acknowledgements

I would like to thank my supervisors, Vic Rayward-Smith and Geoff McKeown, for their help and encouragement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The objective of this thesis is to examine the possible applications of genetic algorithms in cryptology, with the emphasis of the research being in the application of a genetic algorithm in the cryptanalysis of rotor machines with ciphertext only. This thesis covers three large subject matters: statistics, cryptography and genetic algorithms. We present a review of the fundamental ideas of statistical inference, cryptography and genetic algorithms in chapters 2, 3 and 5, which, although by no means comprehensive, provides enough background to understand the techniques applied and to assess the usefulness of the results obtained. Cryptography is the science of conducting secure communication. Cryptanalysis is the study of recovering communications by those other than the intended receiver. Cryptology is the umbrella term for cryptography and cryptanalysis, although cryptography is often used to mean cryptology. Cryptography has attracted much interest in recent years. While historically, cryptography was thought of as of interest to hobbyists and spies only, the increasing need for the secure transmission of

large quantities of information and the advances made in public key cryptography have made cryptography an attractive subject. Chapter 4 describes some of the applications of statistics in cryptology. Statistics forms the basis of much cryptanalysis, and is also important in testing cryptographic systems for possible weaknesses. We describe various statistics used in cryptanalysis and include an iterative technique to find the maximum likelihood estimate of a rotor machine. Chapter 5 looks at genetic algorithms from the perspective of permutation problems. The body of this chapter is a review of some of the operators which have been proposed for the use with permutation problems, with operators being described in the manner used in most of the literature (i.e. informally) and also in terms of permutations. Chapter 6 ties all the ideas together by looking at the applications of genetic algorithms, which utilise the statistical properties of a particular language to find a fitness measure, in the cryptanalysis of rotor machines.

Chapter 2

Statistical Inference

2.1 Introduction

Statistics plays a key role in cryptanalysis. Although statistical analysis alone will rarely give solutions to cryptographic systems, it often plays a central role in a method of attack. The probabilistic variation of plaintext, or possibly of keys, forms the basis of many cryptanalytic techniques, and because of this an understanding of the fundamental principles of statistics is vital. This chapter describes the underlying assumptions and principle techniques of the two basic approaches to statistical inference, classical and Bayesian inference. [77] provides a definitive coverage of the theory of statistics, while [5] presents a good comparison of the different approaches to statistical inference.

It is useful to attempt to clarify what is meant by statistical inference. In the most general terms statistics can be considered the study of numeric properties of populations, i.e. groups of particular objects, such as the height

of men in England. A population may also be generated by a particular process, for example dice rolling generates the population of all dice rolls, or sampling letters generates the population of all letter combinations. More specifically, Statistics can reasonably be defined as the "the study of how information should be employed to reflect on, and give action guidance in, practical situations involving uncertainty." [5].

Statistical inference is the process of making statements about the properties of a population based on a sample of possible observations and any other available information. An inferential process must

1. construct some mathematical model of situations involving uncertainty,
2. define what is meant by information and
3. somehow clarify the difference between action guidance and reflection

before being able to say anything useful about a population. The first section describes some of the basic mathematical tools used in statistics and the different interpretations used in the modelling of uncertainty. The second briefly defines different ideas of what constitutes information and the last section describes what is meant by action guidance.

2.2 Uncertainty

A practical situation involving uncertainty is one where there is more than one possible outcome, and the outcome is indeterminate. The first requirement of any inferential process is a formalisation of this idea of uncertainty by defining the concept of probability. This must be expressed logically and

mathematically so that the methods of probability theory or probability calculus, reached through logical deduction, can be used to obtain a description of the probabilistic properties of the data. The aim of statistical inference is to utilise the deductive link of probability theory to make inferences about the model of the real world situation using information provided from events that have arisen in that situation. A definition of probability which is used when making statements about the external world must be consistent with the axioms from which probability theory is derived.

2.2.1 Rules of Probability

Probability theory is developed using a set theoretic framework. A statistical experiment is the process of observation or measurement of a member of a population. The universal set is the set of all possible outcomes of a statistical experiment, the *sample space*, Ω. Sample spaces may be sets of finite magnitude, as for the experiment of rolling a die, where $\Omega = \{1, 2, 3, 4, 5, 6\}$ or of infinite cardinality. Sample spaces of infinite size are further distinguished between countable and uncountable sets.

Countably infinite sets are those which can be one-to-one mapped to the set of integers. For example the experiment of rolling a dice until a six comes up has the infinitely countable sample space $\Omega = \{< 6 >, < 1\ 6 >, < 2\ 6 > , \ldots, < 1\ 1\ 6 >, \ldots\}$. Sample spaces with finite or countably infinite sample spaces are said to be *discrete*. Some experiments have sample spaces which are neither finite nor countably infinite. The outcome can be measured to any desired degree of accuracy. For example experiments where the outcome is a measure of time, temperature or distance have sample spaces with an

infinite number of elements which cannot be mapped to the integers. These sample spaces are said to be *continuous*.

An *event*, A, is a subset of outcomes such that $A \subseteq \Omega$. Event A occurs if the experiment results in one of the outcomes in A.

Axioms

1. To every random event, A, there corresponds a number, $P(A)$, the probability of A, which satisfies $0 \leq P(A) \leq 1$.

2. The probability of the certain event is unity, $P(\Omega) = 1$.

3. If there exists some countable set of events $A_1, A_2, \ldots$
   then $P(A_1 \cup A_2 \cup \ldots) = P(A_1) + P(A_2) + \cdots$ if $A_i \cap A_j = \emptyset \ , \forall i \neq j$
   (i.e. if all events are mutually exclusive.)

All the rules of probability can be derived from these three axioms.

Conditional Probability

It is often the case that the sample space can be specified in a number of ways. If we are concerned with the probability of any letter taken randomly from a string of text being an e, then obviously this probability will depend on the string of text of interest.

The *conditional probability* of event A relative to sample space S is denoted $P(A|S)$. In conditioning the set of outcomes is restricted to some subset $B \subset \Omega$, in effect a new sample space. For example, if event A is a subset of all five letter words in a certain language, then a possible conditioning is the subset of all five letter words beginning with the letter b. The

conditional probability for A given B, that is the probability of the outcome being in A given that the outcome must be in B, is defined as

$$P(A|B) = \frac{P(A \cap B)}{P(B)},$$

And by symmetry

$$P(A \cap B) = P(A|B)P(B) = P(B|A)P(A).$$

Two events, A and B, are *independent* if $P(B|A) = P(B)$ and $P(A|B) = P(A)$, that is to say if the occurrence of one does not effect the probability of the other occuring. So two events are independent iff

$$P(A \cap B) = P(A)P(B).$$

Bayes' Theorem

It is often the case that the outcome of an experiment depends on what happens in intermediate stages, i.e. the final outcome is conditional on some previous outcome. Bayes' theorem allows the calculation of these inverse probabilities under the assumption that all the conditional probabilities are known. If $B_1, \ldots, B_n$ constitute a partition of the sample space Ω and $P(B_i) \neq 0$ for i=1,...,n, then for any event A in Ω such that $P(A) \neq 0$

$$P(B_r|A) = \frac{P(B_r)P(A|B_r)}{\sum_{i=1}^{n} P(B_i)P(A|B_i)}.$$

The above definitions are all that is necessary to develop the calculus of probability, but do not provide a means of relating probability statements to real world situations or offer any way to evaluate probabilities for specific events. That requires a definition of the meaning and method of evaluation of

probability statements. Cohen [13] discusses the philosophical ramifications of the various probability definitions. The main two probability paradigms are discussed below.

2.2.2 Frequency Probability

The frequentist view is that probability is only quantifiable in statistical experiments which are potentially infinitely repeatable, and where repetitions of the experiment are independent. The probability of event A observed N_A times from N observations is then defined to be

$$lim_{N\to\infty}\frac{N_A}{N}.$$

It is important to note that N_A/N is not a sequence for which it is possible to derive the limit analytically, but rather the ratio of successes (occurrences of A) against trials from a succession of empirical observations, which seems to tend to a number as N increases, called the probability of event A. It should also be apparent that the acceptance that a situation fulfils the requirements of independence and repeatability is subjective, and that many situations where probability statements may be useful do not satisfy the criteria.

2.2.3 Subjective Probability

The problem with the frequentist definition is that there are many situations where it is not applicable, and the method of deciding when a situation fulfils the necessary criteria is somewhat vague. To overcome this problem various alternative definitions of probability have been suggested, the most successful of which has been subjective probability. Subjectivists consider probability

to be a measure of the degree of belief of an individual in the occurrence of an event based on their relevant experience. All probabilities are conditional, given personal knowledge H and event A, $p(A|H)$ is the subjective probability of A. Observation modifies our opinion, and no objective probability exists for an event. The advantage of this approach is that it allows probability statements about hypotheses. For some hypothesis, H, about the population and a realisation of an experiment, event A, the so called "inverse probability", $P(H|A)$, has no real interpretation under a frequentist definition. For example the statement "the probability that the mean of the distribution of some population is between 2 and 3 is 0.75 given our observation" has no real interpretation under a frequency definition, which, as we shall see later, relies on sampling distributions. Under the subjective definition however it expresses the individual's degree of belief in the hypothesis given the evidence of observation. The problems arise in actually quantifying subjective probabilities. Schemes that derive probabilities in terms of betting odds under rational restrictions have been suggested, but many still find them unsatisfactory. [68, 47, 16] discuss subjective probabilities in detail. In practice most statisticians do not worry about these fundamental difficulties and are flexible. If there is enough evidence to support the use of inverse probabilities then they use them, as we shall see later in the section on Bayesian inference.

2.3 Modelling Uncertainty

2.3.1 Functions and Variables

Once probability has been defined, it is necessary to outline how it is utilised in describing the properties of the population in which we are interested. If Ω is a sample space (with a probability measure) and X is a real-valued function defined over the elements of Ω,

$$X : \Omega \rightarrow \Re,$$

then X is called a *random variable*. A random variable is a function that describes the statistical experiment. With reference to the previous example, if X is a random variable defined over the set of all dice rolls representing the number of rolls required to observe a six, then if the observed number of rolls required is 2 we write $X = 2$ for the subset, S, of the sample space, Ω, where

$$S = \{< 1, 6 >, < 2, 6 >, < 3, 6 >, < 4, 6 >, < 5, 6 >\}.$$

So generally $X = x$ should be interpreted as the subset of Ω for which the random variable X takes on the value x (X denotes the random variable and x a particular realisation of X). The probability that the experiment results in outcome assigned value x is denoted

$$P(X = x).$$

Distribution Functions

A random variable is discrete if the associated sample space is discrete, and continuous otherwise.

If X is a discrete random variable the function given by

$$f(x) = P(X = x)$$

for each x within the range of X is called the *probability distribution* of X.

If X is a continuous random variable a function f with values $f(x)$ is called a *probability density function* (pdf) of the random variable X iff

$$P(a \leq X \leq b) = \int_a^b f(x)dx$$

for any real constants a and b with $a \leq b$.

Multivariate Distributions

It is clear that several random variables can be defined over the same sample space. An experiment concerned with the population of students at UEA, for example, may be performed to find out about various characteristics such as height, intelligence or age. These characteristics may be related, in which case it is of interest to look at the joint probability distribution.

If $X_1, X_2, \ldots, X_n$ are n discrete random variables the function given by

$$\begin{aligned} f(x_1, x_2, \ldots, x_n) &= P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) \\ &= P(X_1 = x_1 \cap X_2 = x_2 \cap \cdots \cap X_n = x_n) \end{aligned}$$

for each set of n of values $(x_1, x_2, \ldots, x_n)$ within the range of $X_1, X_2, \ldots, X_n$ is called the *joint probability distribution* of $X_1, X_2, \ldots, X_n$.

If $X_1, X_2, \ldots, X_n$ are continuous random variables the function given by

$$P(X_1, X_2, \ldots, X_n \in A) = \int_A \ldots \int f(x_1, x_2, \ldots, x_n)dx_1dx_2 \ldots dx_n$$

for any region $A \subset \Re^n$ is called the joint probability density function of $X_1, X_2, \ldots, X_n$.

Marginal Distributions

Although joint distributions are useful, it is sometimes desirable to consider the distribution of one variable only. This is called the marginal distribution. The two variable case described below can easily be extended to larger number of variables.

If X and Y are discrete random variables and $f(x, y)$ is their joint probability distribution then the function given by

$$g(x) = \sum_y f(x, y)$$

for all x in the range of X is called the marginal distribution on X. If X and Y are continuous random variables and $f(x, y)$ is their joint pdf then the function given by

$$g(x) = \int_{-\infty}^{\infty} f(x, y)dy \text{ for } -\infty < x < \infty$$

is called the marginal density of X.

Conditional Distributions

If $f(x, y)$ is the value of the joint probability distribution of discrete random variables X and Y at (x, y) and $h(y)$ is the value of the marginal distribution of Y at y then the function given by

$$f(x|y) = \frac{f(x, y)}{h(y)} \quad h(y) \neq 0$$

is called the *conditional distribution* of X given $Y = y$.

If $f(x, y)$ is value of the the joint probability distribution of continuous random variables X and Y at (x, y) and $h(y)$ is the value of the marginal density of Y at y then the function given by

$$f(x|y) = \frac{f(x,y)}{h(y)}, \quad h(y) \neq 0 \quad for -\infty < x < \infty$$

is called the *conditional density* of X given $Y = y$.

The aim of statistical inference is then to attempt to fully describe the pdf of a population from sample data and other relevant information. This process is aided by the fact that many distributions can be described by summary measures of location and dispersion such as the arithmetic mean and variance. These are called *parameters*.

2.3.2 Expectation and Moments

Mathematical expectation is important in statistics as it allows us to describe distributions by simple summary measures. The expectation of a random variable, termed its expected value, is defined with the use of moments. The rth moment about the origin of a random variable, X, denoted μ'_r, is the expected value of X^r.

$$\mu'_r = E(X^r) = \Sigma x^r . f(x),$$

for $r = 0, 1, 2, \ldots$ when X is discrete and

$$\mu'_r = E(X^r) = \int_{-\infty}^{\infty} x^r . f(x) dx.$$

μ'_1 is called the *mean* of the distribution of X and denoted μ,

$$\mu = \mu'_1 = E(X).$$

The r^{th} moment about the mean of a random variable, X, denoted μ_r, is the expected value of $(X-\mu)^r$.

$$\mu_r = E((X-\mu)^r) = \Sigma(x-\mu)^r.f(x),$$

for $r = 0, 1, 2, \ldots$ when X is discrete and

$$\mu_r = E((X-\mu)^r) = \int_{-\infty}^{\infty}(x-\mu)^r.f(x)dx$$

when X is continuous. μ_2 is called the *variance* of the distribution of X and is denoted σ^2,

$$\sigma^2 = \mu_2 = E((X-\mu)^2).$$

σ^2 can be derived from the first and second moment about the origin.

$$\begin{aligned}\sigma^2 &= \mu_2' - (\mu_1')^2 \\ &= E(X^2) - (E(X))^2\end{aligned}$$

The positive square root of the variance, σ, is called the standard deviation. The mean and variance of a random variable play a central role in statistical inference.

2.3.3 Sampling Distributions

As already said, statistical experiment is concerned with predicting the results of chance outcomes. In the finite case these constitute a subset or sample of observations from the population. In the infinite case they tend to be observations of identically distributed random variables, the distribution of which is called the population distribution. For example a ten letter string sampled from the a certain page in a book is a subset of the set of all

ten letter strings on that page, but a ten letter string generally could be described as 10 realisations of the identically distributed random variables with population distribution given by a single letter frequency distribution table or 5 realisations of the identically distributed random variables with population distribution given by a two letter frequency distribution table, or indeed any other length of n-gram distribution. Most methods of inference are concerned with random samples from infinite populations. If $X_1, X_2, \ldots, X_n$ are independent and identically distributed random variables we say that the constitute a random sample from the infinite population given by their common distribution $f(X)$, and their joint distribution at $x_1, x_2, \ldots, x_n$ is given by

$$f(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} f(x_i).$$

Random variables which are functions of a set of random variables $X_1, X_2, \ldots, X_n$ constituting a random sample are called *statistics*. Generally the statistics of primary concern are the *sample mean* and the *sample variance*. If $X_1, X_2, \ldots, X_n$ constitute a random sample, then

$$\bar{X} = \frac{\sum_{i=1}^{n} X_i}{n}$$

is called the *sample mean* and

$$S^2 = \frac{\sum_{i=1}^{n} (X_i - \bar{X})^2}{n-1}$$

is called the *sample variance*.

Sampling Distribution of the Mean

Since statistics are random variables they will follow a probability distribution. These are called sampling distributions. The sampling distribution

of the mean plays an important role in inference. If $X_1, X_2, \ldots, X_n$ constitute a random sample from an infinite population with mean μ and variance σ^2, then

$$E(\bar{X}) = \mu \quad \text{and} \quad var(\bar{X}) = \frac{\sigma^2}{n}.$$

Central Limit Theorem

If $X_1, X_2, \ldots, X_n$ constitute a random sample from an infinite population with mean μ and variance σ^2, then the limiting distribution of

$$Z = \frac{\bar{X} - \mu}{\frac{\sigma}{\sqrt{n}}}$$

as $n \to \infty$ is the standard normal distribution.

The mean $\bar{X}$ tends to be distributed normally (see below) about μ with variance σ/n. The central limit theorem justifies approximating the distribution of $\bar{X}$ with a normal distribution mean μ and variance $\frac{\sigma^2}{n}$ for large n, with $n > 30$ generally being sufficient.

2.3.4 Some Common Probability Distributions

Discrete Distributions

Bernoulli

A random variable, X, has a *Bernoulli distribution* iff its probability distribution is given by

$$f(x;\theta) = \theta^x(1-\theta)^{1-x} \quad \text{for} x = 0, 1.$$

An experiment to which the Bernoulli distribution applies is known as a *trial*.

Binomial

A random variable, X, has a *binomial distribution* iff its probability distribution is given by

$$b(x; n, \theta) = C_x^n \theta^x (1-\theta)^{n-x} \text{ for } x = 0, 1, \ldots, n.$$

Poisson

A random variable X has a *Poisson distribution* iff its probability distribution is given by

$$p(x; \lambda) = \frac{\lambda^x e^{-\lambda}}{x!} \text{ for } x = 0, 1, 2, \ldots, n.$$

The Poisson distribution is commonly used as an approximation to the binomial. If n is large ($n > 20$) and θ is small ($\theta < 0.05$), the Poisson distribution with $\lambda = n\theta$ provides a good approximation of the binomial. For example suppose in the above example we are interested in the occurrences of b and we know $P(X = b) = 0.01$. Using the Poisson approximation with $\lambda = n\theta = 1$

$$P(X = 0) = \frac{1^0 e^{-1}}{0!} = 0.3679.$$

and

$$P(X = 3) = \frac{1^3 e^{-1}}{3!} = 0.0613.$$

Continuous Distributions

Normal

A random variable X has a *normal distribution* iff its probability distribution is given by

$$n(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \quad \text{for} -\infty < x < \infty$$

where $\sigma > 0$.
the normal distribution is particularly important as many natural phenomena are distributed normally.

Gamma

A random variable X has a *gamma distribution* iff its probability distribution is given by

$$f(x) = \begin{cases} \frac{1}{\beta^\alpha \Gamma(\alpha)} x^{\alpha-1} e^{-\frac{x}{\beta}} & \text{for } x > 0 \\ 0 & \text{elsewhere,} \end{cases},$$

where

$$\Gamma(\alpha) = (\alpha - 1)!.$$

2.4 Information

Within a statistical inference framework, information is assumed to take one of three forms:

1. *Sample data*, a set of observed outcomes from a situation assumed to be independent repetitions under identical circumstances, i.e. values of

identically distributed random variables, whose distribution is referred to as the *population distribution*. It is the only form of information formily abmissible within the classical inference framework.

2. *Prior information*, the information available to the experimenter about the situation before taking sample data. This may be based on previous observations of similar situations or personal knowledge or opinion about the probability distribution of the random variable in question. Crucial to the methods of Bayesian inference, prior information is generally dismissed by followers of the classical approach because of its generally subjective nature.

3. *Future consequences* of actions resulting from any inferences made, central to the methods of decision theory. Quantifying consequences often requires some subjective element and decision theory tends to fit into the inferential model better as an extension of Bayesian inference.

2.5 Action Guidance

The traditional view is that statistical inference is purely a branch of applied mathematics and, as such, should not involve such subjective elements as decision making and consequences. Such things should be left to the people requiring the statistics, not the statistician. The more modern attitude is that it is inevitable that the information will be used this way, so surely it is better to formally incorporate it within the inferential process rather than

ignore it or, perhaps worse, only consider it informally through hypothesis testing.

2.6 Classical Inference

Considered the mainstream of statistical practice, classical inference takes the frequentist probability paradigm as its basis. It admits sample data as the only (formally) admissible information, which it hopes to utilise to find out information about the sampling distribution.

A random variable, X, is defined on sample space, Ω, the set of all possible outcomes of X. The inherent variablility of X over Ω is described by a pdf, $p_\theta(X)$, which is assumed known except for parameter(s), θ, defined on parameter space, Θ.

Given sample data, $x_1, x_2, \ldots, x_n$, arising from observation of X, the inferential problem is to determine a value or range of values that can be taken as θ or to include θ. That is to say, identify a subset of the set of possible probability distributions, which contains the correct pdf, $p_\theta(X)$. An *estimator* or *statistic*, T, is a function of the sample data and is itself a random variable. If we use a statistic to estimate θ we cannot guarantee it takes the correct value, so we look to construct statistics that will give good approximations of θ 'in the long run' or 'on average'. Hence the probability distribution or *sampling distribution* of a particular statistic forms the basis of judging the suitability of that statistic in estimating θ.

An intuitively obvious way of constructing an estimator is to calculate T from the sample in exactly the same way as θ is calculated from the popula-

tion. For example, we could use the sample mean to estimate the population mean. We would often expect the sample mean to have 'on average' a value equal to the population mean, and also for the distribution of the sample mean to have less variation about the population mean as the number of observations, n, increases. These intuitive ideas are formalised through the ideas of desirable properties of estimators.

2.6.1 Point Estimators

A point estimator is a function of the data that maps every point in the sample space to one in the parameter space. In order to judge the suitability of estimators, criteria for judging how 'good' they are must be defined. It might also be desirable to find a way of judging whether an estimator is in some way optimal.

Consistency

The statistic T is a consistent estimator of the parameter θ iff for each $c > 0$

$$\lim_{n\to\infty} P(|T - \theta| < c) = 1.$$

If a statistic is consistent with parameter θ it means that the precision of the estimator increases as the sample size increases, and T is said to *converge stochastically* to θ.

Unbiasedness

A statistic T is an unbiased estimator of the parameter θ iff

$$E(T) = \theta.$$

Note that if an estimator is consistent its distribution must, for large samples, have a central value in the neighbourhood of θ. Unbiasedness requires that the estimator has a central value of θ for all sizes of sample.

So the criteria of unbiasedness and consistency formalise our intuitive requirements of an estimator. But it can often be the case that many statistics fulfil both requirements, so further criteria for choosing between estimators are required.

Efficiency

An obvious area of comparison between unbiased estimators is the respective sample variances. It would be expected that the estimator with the smaller variance will be distributed more closely about θ.

Let T_1, T_2, be two unbiased estimators of θ. T_1 is more *efficient* than T_2 if

$$\mathrm{Var}(T_1) < \mathrm{Var}(T_2).$$

The *relative efficiency* of T_2 relative to T_1 is given by

$$\frac{\mathrm{Var}(T_1)}{\mathrm{Var}(T_2)}.$$

Sufficiency

A further, more general, criteria of suitability is *sufficiency*. Suppose we are interested in a statistic, T, which is one of the unlimited possible statistics of θ. If the likelihood function, L, (see below) can be written in the form

$$L(x_1, x_2, \ldots, x_n|\theta) = g(T|\theta)k(x_1, x_2, \ldots, x_n)$$

where $g(T|\theta)$ is a function of T and θ alone and k is independent of θ, then T is called a *sufficient* statistic of θ. Basically this means than any other

statistic we may choose to use would add nothing further to our knowledge of θ.

These properties give methods of comparing one estimator with another, but do not offer a way of judging whether an estimator is optimal. One commonly used criteria for judging optimality is described below.

Minimum Variance Bound Unbiased Estimators (MVBUE)

An important result in the formulation of estimators is that, under certain conditions, there is a lower bound on the variance of an unbiased estimator. If

$$I = -E(\frac{\partial^2 \log L}{\partial \theta^2}),$$

where I is known as the *information function*, then, if T is an unbiased estimator of θ,

$$var(T) \geq \frac{1}{I}.$$

This is known as the *Cramer-Rao inequality*. If

$$var(T) = \frac{1}{I}$$

then T is called the MVBUE of θ. That is to say it will have the lowest variance among all unbiased estimators. Unfortunately there is no guarantee that an MVBUE will exist, but where one does exist it can be said to be optimal.

While the ideas of good criteria and possibly optimal estimators described above give an indication of how we can judge the suitability of estimators, they give no indication of how to actually derive them. We now look at three

commonly used methods of finding estimators.

1. **Method of Maximum Likelihood.**

Likelihood Function.
If $x_1, x_2, \ldots, x_n$ are the values of a random sample from a population with the parameter θ, the *likelihood function* of the sample is given by

$$L(x_1, x_2, \ldots, x_n|\theta) = f(x_1|\theta)f(x_2|\theta)\cdots f(x_n|\theta)$$

which we shall sometimes write as simply L. The likelihood function describes the probability of observing any particular set of sample data given a particular value of θ. The likelihood function plays a crucial role in classical and Bayesian inference and is considered to be the most complete summary of the information provided by the sample data available.

A maximum likelihood estimator (MLE) is one that satisfies the property that the likelihood function, L, is maximum. The method of maximum likelihood is to take as an estimate of θ the value for which the probability of having observed data $\bar{x}$ is maximum. So a MLE, T, is a function which satisfies

$$\frac{\partial L}{\partial \theta} = 0$$

subject to

$$\frac{\partial^2 L}{\partial \theta^2} < 0.$$

A MLE is consistent and has the asymptotic properties of unbiasedness and full efficiency (i.e. $var(T) \to \frac{1}{I}$).

2. **Method of Moments.**

Previously we defined moments for random variables. We now define *sample moments*

The kth sample moment of a set of observations, $x_1, x_2, \ldots, x_n$, m'_k, is

$$m'_k = \frac{\sum_{i=1}^{n} x_i^k}{n}.$$

The method of moments is simply to equate enough moments of the population to moments of the sample to get solvable simultaneous equations in the unknown parameters. So if a population has r different parameters, the method of moments is to solve

$$m'_k = \mu'_k \quad for\ k = 1, 2, \ldots, r.$$

While the method of moments has obvious intuitive appeal, the estimators it produces cannot be said to have any desirable general properties. The method can often serve as a validation of estimators found by other techniques or as a starting point for an iterative maximum likelihood method.

3. **Method of Least Squares.**

If it assumed that our data x arises as an observation of a vector random variable X of size n which are related to the vector of k parameters θ through there expectations by a linear relationship,

$$\begin{aligned} E(X_j) &= p_j(\theta) \\ &= x_{ij}\theta_1 + x_{2j}\theta_2 + \cdots + x_{kj}\theta_k, \end{aligned}$$

then we should choose our estimator $\hat{\theta}$ to minimise the sum of the squared deviation between X_j and their estimated expectation, ie minimise

$$S = \sum_{j=1}^{n} (X_j - p_j(\hat{\theta}))^2.$$

Consider the linear model

$$x = A\theta + \epsilon,$$

where X is an (n $\times$ 1) vector of observations, A is an (n$\times$k) matrix of known coefficients, θ is a (k$\times$1) vector of observations and ϵ is an (n $\times$ 1) vector of error random variables with

$$E(\epsilon) = 0$$

The least squares method is to mimimise the sum of squares

$$S = (x - A\theta)'(x - A\theta),$$

which is minimised when the partial differentials with respect to the elements of θ are zero, giving our estimates

$$\theta = (A'A)^{-1}A'x.$$

2.6.2 Interval Estimation

Point estimates give a point in the parameter space as an estimate of the parameter of interest, but it may be of more use to specify a range of values in which θ lies.

2.6.3 Confidence Interval

The aim is to find two statistics, $T_0,\ T_1$, with the property that parameter θ will lie between the observations of $T_0,\ T_1$ with a probability of $1-\alpha$ for some specified α.

$$P(t_0 \leq \theta \leq t_1) = 1 - \alpha.$$

The interval (t_0, t_1) is called the *confidence interval* for θ while t_0 and t_1 are called the upper and lower confidence limits.

It should be remembered that this confidence interval relates to long term behaviour. A confidence interval should be understood as meaning that, in the long run, $100(1-\alpha)\%$ of the intervals created using statistics $T_0,\ T_1$, will contain the true parameter value θ. A wrong interpretation is that the probability that the parameter value is in range $[t_0, t_1]$ is $1-\alpha$. With a frequentist definition of probability this statement has no meaning.

2.6.4 Hypothesis Testing

A hypothesis in this context means a statement or conjecture about the distribution of one or more random variables. For parameterised distributions this takes the form of a statement that the true parameter value lies in some subset of the parameter space.

A statistical hypothesis test addresses the question of whether the observed data is consistent with a particular hypothesis by partitioning the sample space into two exhaustive subsets, members of which are deemed respectively 'consistent with hypothesis' and 'not consistent with hypothesis'. That is to say, testing the hypothesis $\theta \in \delta$ where $\delta \subset \Omega$ involves creating a

partion of sample space, Ω, $\{S_0, S_1\}$, so that if $x \in S_0$ we accept that $\theta \in \delta$ and if $x \in S_1$ we accept that $\theta \in \Theta - \delta$. S_0 is known as the *acceptance region* and S_1 as the *critical region* of the test.

If a hypothesis completely specifies the distribution in question, including all parameter values, it is termed *simple*. If it does not it is a *composite* hypothesis. Geometrically, a simple hypothesis specifies a point in parameter space while a composite hypothesis specifies a subregion.

The hypothesis under test, $\theta \in \delta$, is called *the null hypothesis*, H_0, and the specified alternative, $\theta \in \Theta - \delta$, is *the alternative hypothesis*, H_1.

The partioning of the sample space is achieved by means of a *test statistic* which will tell the tester for each possible outcome what action to take. A *test of significance* rules that H_0 is accepted if the observed value of test statistic T, t, lies in a certain range. Otherwise H_0 is rejected. H_0 is accepted unless the observed data has a sufficiently small probability of occurring when H_0 is true. So a hypothesis is accepted unless there is sufficient evidence to reject it. The probability of rejecting H_0 when H_0 is in fact true (denoted Type I error) is called the *significance level*, α.

$$P(\text{Type I error}) = \alpha = P(X \in S_1 | \theta \in \delta).$$

An alternative type of error, *Type II error*, is that of accepting H_0 when H_1 is true. The probability of type II error is denoted β.

$$P(\text{Type II error}) = \beta = P(X \in S_0 | \theta \in \Theta - \delta)$$

or

$$1 - \beta = P(X \in S_1 | \theta \in \Theta - \delta).$$

$1 - \beta$ is called the *power*, p, of the test.

The only restriction so far on the critical region is that it has size α, but obviously there could be many different critical regions for any particular test. An obvious criteria for choosing a critical region is the probability of type II error, β, or equivalently the power, p. A critical region for testing a simple null hypothesis whose power is at least as large as any other region at the same significance level is called a *best critical region* (BCR) and the associated test a *most powerful* (MP) test. For testing a simple hypothesis against a simple hypothesis finding a BCR of size α is simply a matter of satisfying the following lemma.

Neyman Pearson Lemma
If C is a critical region of size α and k_α is a constant such that

$$\frac{L(x|H_0)}{L(x|H_1)}) \leq k_\alpha \text{ when } x \text{ is in } C$$

and

$$\frac{L(x|H_0)}{L(x|H_1)}) \geq k_\alpha \text{ when } x \text{ is outside } C$$

then C is a BCR for testing H_0 against H_1.

For composite hypthesis tests a method closely connected to the maximum likelihood method of estimation is used to construct a critical region, called *likelihood ratio tests*. Let L' be the maximum likelihood estimate of θ over the entire parameter space, and let L'_0 be the maximum value of the likelihood function for the values of θ in δ. The ratio of these two random

variables,

$$\lambda = \frac{L_0'}{L'},$$

is referred to as a value of the *likelihood ratio statistic*.

2.7 Bayesian Inference

Bayesian inference utilises Bayes' theorem, extended to include currently available information, to modify opinion by experience. [53, 63] give good introductions to Bayesian inference.

Extended Bayes' Theorem

Let $B_1, B_2, \ldots, B_n$ constitute a partition of the sample space Ω and let H be the information currently available then for any event A it follows from our definitions of conditional probability that

$$P(B_i, A|H) = P(A|H)P(B_i|A, H) = P(B_i|H)P(A|B_i, H).$$

Thus,

$$\begin{aligned} P(B_i|A, H) &= \frac{P(B_i|H)P(A|B_i, H)}{P(A|H)} \\ &= \frac{P(B_i, A|H)}{\Sigma_i^n P(B_i, A|H)}. \end{aligned}$$

The theorem gives the probabilities of the events B_i when A is known to have occured. Probabilities of the type $P(B_i|A, H)$ are called *posterior probabilities*, those of type $P(B_i|H)$ are the *prior probabilities* and those of type $P(A|B_i, H)$ are the *likelihood*. The relationship between these probabilities can be summarised as

$$\text{posterior} \propto \text{prior} \times \text{likelihood}.$$

Bayes' theorem provides us with a way of assigning a probability value to a particular hypothesis for a given set of data.

Firstly we will discuss the methods of estimation and hypothesis testing for parameterised systems. A random variable, X, defined on a sample space, Ω, is assumed to have a parameterised pdf, $p_\theta(X)$, known except for parameter(s) θ. The aim of the inferential procedure is to choose one of the family of probability distributions, $p_\theta(X)$, i.e. select an 'optimum' value of θ, based on prior information and sample data. The prior degree of belief in possible values of θ are assumed to be fully described by the *prior distribution*, $\pi(\theta)$. Sample data, expressed as the likelihood function $L(x|\theta)$, modifies our opinion about the probability distribution of θ by the application of Bayes' theorem, to produce the *posterior distribution*, which now fully describes our degree of belief about values of θ.

$$\pi(\theta|x) = \frac{\pi(\theta)L(x|\theta)}{\int_\Omega \pi(\theta)L(x|\theta)d\theta}.$$

2.7.1 Point Estimators

A Bayesian point estimator of parameter θ, $\hat{\theta}$, is simply the value that maximises the posterior distribution $\pi(\theta|x)$. It can be interpreted (with a subjective definition of probability) as the most probable value of θ given the present information available.

2.7.2 Bayesian Confidence Regions (Credible Regions)

A region S_α is a $100(1-\alpha)$ credible region if

$$\int_{S_\alpha} \pi(\theta|x)d\theta = 1-\alpha.$$

The problem, similar to that encountered in classical inference, is that there may be any number of regions containing a $(1-\alpha)$ proportion of the posterior distribution. However it seems reasonable to choose S_α so that S_α does not exclude any value of θ more probable than any value of θ already in S_α.

2.7.3 Bayesia hypothesis Tests

The Bayesian approach of calculating 'inverse' probabilities makes it possible to have a direct evaluation of the probability of two hypotheses. There is no need to make a distinction between the null and the alternative hypothesis. Suppose the two hypotheses are $H : \theta \in \delta$ and $H' : \theta \in \Theta - \delta$, then the posterior degrees of belief in the hypotheses are

$$P(H|x) = \int_H \pi(\theta|x)d\theta = 1 - P(H'|x).$$

We can simply quote $P(H|x)$ as the critical level of H or assign a significance level and make a decision about H if so required.

2.7.4 Specifying a Prior

If a subjective definition of probability is accepted then the process of Bayesian inference is much simpler and intuitive than that of classical inference. The difficulty is the practical problem of specification of a prior distribution. This

problem is best approached from the point of view of how much prior knowledge we possess.

Prior Ignorance

In some situations we may have no prior knowledge, i.e. no particular belief about the values the parameter may take. In such cases it is common to call on the *Bayes-Laplace principle of insufficient reason*, which states that ignorance implies that the probabilies of each possible value are equal. If the parameter space is unbounded, for example the set of real numbers, then the probability assignment will be improper, i.e. the probability density will be greater than one over some interval. In practice this is not a problem as the posterior distribution is always normalised.

Vague Prior Knowledge

The relative importance of the prior and the likelihood function is judged by the contribution they make to the posterior distribution. If the prior distribution is closer to a uniform distribution than the likelihood function then the posterior distribution will be close to the normalised likelihood function. In other words the sample data provides information that overwhelms our initial opinion. In this case we are said to have *vague prior knowledge* and the behaviour of the posterior is governed by the *principle of precise measurement*. This states that for large enough samples the posterior is robust under any prior. As the sample size increases the posterior becomes more dominated by the likelihood function. This principle is an important point in the defence of Bayesian inference, in that the most common critisism re-

lates to the arbitrary nature of subjective priors. The principle of precise measurement means that personal prior distributions will converge with a suitable quantity of observation.

Substantial Prior Knowledge

When the posterior distribution departs significantly in form from the likelihood function the prior information is said to be substantial. Any inferences made from the posterior will depend more on the form of the prior than on the observed data. In such cases specification of the prior becomes of paramount importance.

2.8 Decision Theory

Decision theory is concerned with situations where it is necessary to make a decision about about the appropriate course of action based on the information available. In order to decide on a course of action it is necessary to incorporate *consequences* of actions into the inferential model. Let the *action space*, A, be partitioned into n distinct possible actions $(a_1, a_2, \ldots, a_n)$. Let a parameter, θ, be defined on parameter space Θ. Let the *loss function*, $l(a, \theta)$, be defined on $A \times \Theta$. Decision theory aims to suggest a method for choosing an optimal action with respect to the loss function and other available information about the parameter value.

2.8.1 Loss Function

The loss function, $l(a, \theta)$, measures the loss which arises if action a is taken for any particular parameter value $\theta \in \Theta$. Defining a loss function can be difficult. Most often the loss function is defined in terms of some monetary unit, so that $l(a, \theta)$ reflects the *cost* of action a if the parameter, or *state of nature* is θ. This approach is fine if the situation allows such an 'objective' quantification of cost, and supporters of the classical approach maintain that the mathematical decision guidance process is only valid in such situations. Those who support a subjective interpretation of probability define the loss function in terms of *Utility*.

Utility

The concept of utility is based on the observation that an individual inevitably chooses between courses of action and in doing so expresses some ordering of preference between actions. Utilities are numerical values which express the degree of preference between consequences within a logical framework of behaviour. The idea of utilities is closely tied to that of subjective probabilities. For our purposes it is enough to say that to utilise decision theory a loss function must be defined, and whether it is valid to do so is a question fundamental to the particular inferential paradigm adopted. For a more detailed discussion of utility see [47].

2.8.2 Defining Best Action

Using Sample Data Alone

The aim of decision theory is to identify a *decision function*, δ, that specifies the action to be taken when x is observed, which somehow optimally minimises *expected loss* in comparison to other possible decision functions. The consequence of a decision is to suffer a loss, $l(\delta(x), \theta)$, when the parameter value is θ. The expected loss in described by the *risk function*, R,

$$R(\delta, \theta) = \int_{\Omega} l(\delta(x), \theta) f(x|\theta) dx.$$

An obviously good decision function would be one that had uniformly minimium risk for all values of θ. Unfortunately such functions are rarely available, and some method of choosing a decision function is necessary. The first step is to restrict the class of decision functions. A decision rule, δ, is *admissible* if there is no rule δ' such that

$$R(\delta', \theta) \leq R(\delta, \theta) \text{ for all } \theta.$$

i.e. a decision rule is admissible if there is no other decision rule which is has lower expected loss for all possible values of θ. One possible way for choosing amongst the set of admissible decision rules is to use the *minimax* criteria, which suggests choosing the decision function for which the maximum risk over the parameter space is a minimum.

$$\min_{\delta} \max_{\theta} R(\delta, \theta)$$

This is generally an extremely cautious decision rule. It corresponds to ensuring that the worst that can happen is as good as possible.

Using Prior Information and Sample Data

The use of critria such as minimax is as much as can be achieved without prior information. If a Bayesian approach is followed the problem of choosing a decision function is much simplified. The *Bayes' decision rule* is to minimise the expected risk

$$\begin{aligned} E(R) &= \int R(\delta,\theta)\pi(\theta)d\theta, \\ &= \int\int l(\delta,\theta)f(x|\theta)\pi(\theta)dxd\theta. \end{aligned}$$

The minimum of $E(R)$ is called the *Bayes' risk*. This provides a means of finding an optimum decision for a particular prior.

So far this chapter has covered the broad issues of statistical inference. The final section looks at one specific topic, important in cryptanalysis, which highlights the problem of defining a sample space in a situation involving uncertainty, namely the problem of modelling a language.

2.9 Modelling a Language

Suppose we are interested in providing a probabilistic model of a language with m letters.

Single Letter Frequencies

We can model a language by defining a string of n letters as n independent observations of a random variable, X, defined on the set of all letters, with range Z_m.

To estimate the probability distribution of X we take a large random sample, of size N. If N_i is the number of occurrences of the ith letter, then we estimate $P(X = i)$ with the sample relative frequency,

$$P(X = i) = \frac{N_i}{N} \quad \text{for} \quad i = 0, 1, \ldots, m-1.$$

Two Letter Frequencies

Let random variable, X, be defined on the sample space of all two letter combinations with range $Z_m \times Z_m$.

Plaintext, $\mathbf{x}=(x_0, x_1, \ldots, x_n)$, can then be modelled as $\frac{n}{2}$ independent observations of X, where the probabilities

$$P(X = i) \quad \text{for} \quad i = 0, 1, \ldots, mm-1,$$

are again estimated from observation of the language of interest.

Markov Chains

Plaintext $\mathbf{x}$ is again seen as n independent observations of a random variable, X, defined on the set of letters, with range Z_m. However, the probabilities of observing any element of Z_m are now defined conditional on the previously observed letter. Let T be an $m \times m$ *transition matrix* with elements

$$T_{st} = P(X_j = s | X_{j-1} = t) \;\; 0 \leq s, t < m, \; j > 0$$

Then the probability of observing plaintext $\mathbf{x}$ is defined as

$$P(\mathbf{x}) = h(x_0)P(x_1|x_0)P(x_2|x_1)\cdots P(x_{n-1}|x_{n-2}),$$

where

$$h = (h(0), h(1), \cdots, h(m-1))$$

is the *equilibrium distribution* of X_0, found by solving the linear system of m equations given by

$$\begin{aligned}
h(0) &= h(0)P(0|0) + \cdots + h(m-1)P(0|m-1) \\
h(1) &= h(0)P(1|0) + \cdots + h(m-1)P(1|m-1) \\
\vdots &\quad\; \vdots \\
h(m-1) &= h(0)P(m-1|0) + \cdots \\
&\quad\; + h(m-1)P(m-1|m-1)
\end{aligned}$$

The conditional probabilities of T are again estimated by the relative frequency of a sample of observed text.

More complex models, for example based on the probabilities conditional on the previous two letters, can easily be defined.

Chapter 3

Cryptography and Cryptanalysis

3.1 Background

Cryptography is the science of making communications unintelligible to everyone except the intended receiver(s). A cryptosystem is a set of algorithms, indexed by some key, for encoding messages into ciphertext and decoding them back into plaintext. Cryptanalysis is the attempt to recover the plaintext without knowledge of the key. This section mathematically describes a general cryptographic system, the second section gives the ground rules for cryptanalysis and for measuring the security of cryptosystems and later sections give details of the more common cryptosystems that are in use now or have been used in the recent past. Cryptographic systems can be divided into those using a secret key, called *symmetric cryptosystems*, and those using

a public/private pair of keys, called *public key or asymmetric cryptosystems*. The model for a secret key system, first proposed by Shannon [72], is shown in figure 3.1. The model for a public key system is given in figure 3.2.

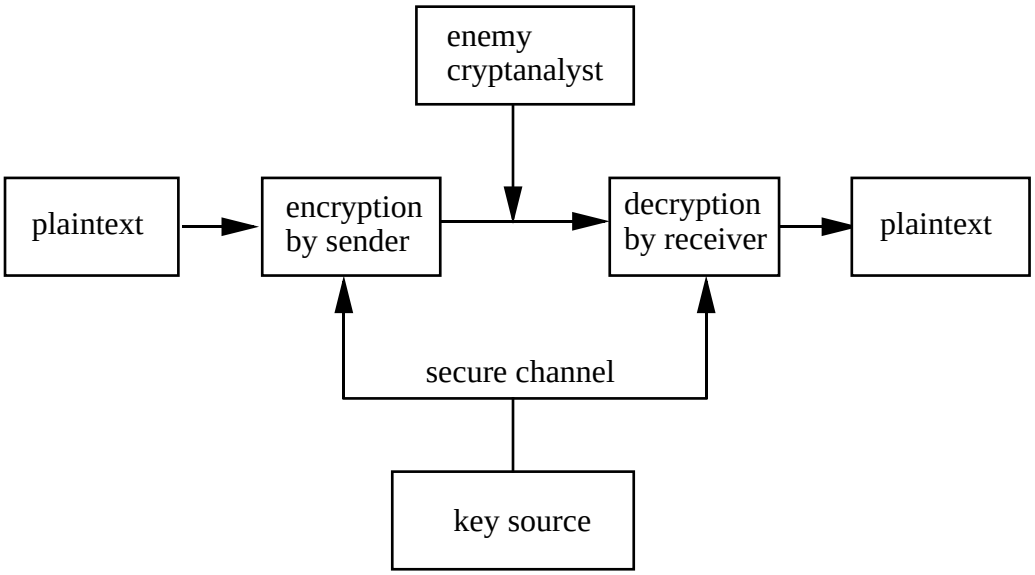

Figure 3.1: Shannons model of secret communication

We describe a cryptosystem in terms of functions defined on strings of text.

3.1.1 Description of Cryptosystems

Set Theory of Strings

Let Σ represent a nonempty ordered finite set of symbols, called an alphabet.

For any alphabet, Σ, let

$$\Sigma^1 = \Sigma,$$

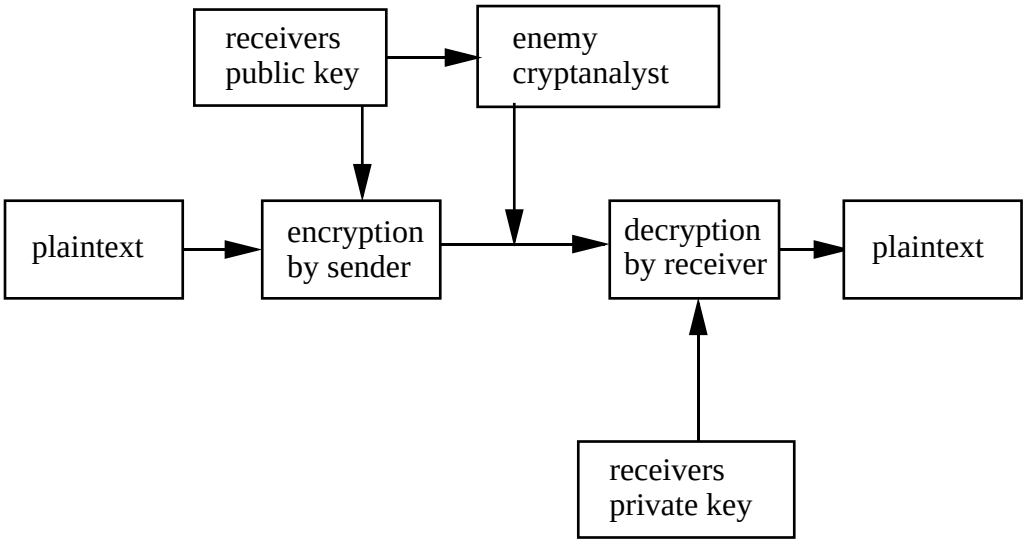

Figure 3.2: A simple model of a public key cryptosystem

and

$$\Sigma^{n+1} = \{xy \mid x \in \Sigma, y \in \Sigma^n\},$$

where xy denotes the juxtaposition of x and y. A string, then, is an element of the set

$$\Sigma^+ = \bigcup_{n=1}^{\infty} \Sigma^n.$$

Let Σ_1 be the alphabet, of size q_1, consisting of the elements of Z_{q_1}, which represents the possible letters of plaintext for a cryptosystem. Let Σ_2 be the alphabet, of size q_2, consisting of the elements of Z_{q_2}, which represents the possible letters of ciphertext for a cryptosystem.

An alphabet for a cryptographic system is not necessarily the same as an alphabet for a language. For example, a plaintext message HELP may, for some systems, represent a four letter message from alphabet Z_{26}. Alternatively, if the message is written in binary, with $A = 00000$ to $Z = 11001$ (and

assuming 11010 to 11111 are also valid symbols), a cryptosystem could be defined with alphabet Z_2, in which case the message is 20 letters long, or, at the other extreme, with alphabet $Z_{2^{20}}$, giving a message of one letter.

The plaintext message, x, and the ciphertext, y, are strings, such that

$$x \in \Sigma_1^+ \quad \text{and} \quad y \in \Sigma_2^+.$$

A *key*, k, for a cryptographic system is a small piece of information defined on a *keyspace*, K.

Definition 3.1 *A substitution on Σ_1, with codomain Σ_2, is a one-to-one function, s,*

$$s : \Sigma_1 \quad \rightarrow \quad \Sigma_2$$

Let S be the set of all substitutions from Σ_1 to Σ_2.

Definition 3.2 *A substitution enciphering function, e, is a one-to-one function*

$$e : \Sigma_1^+ \quad \rightarrow \quad \Sigma_2^+$$

where the the i^{th} plaintext letter is enciphered into the i^{th} ciphertext letter by means of a substitution.

This means that an enciphering function can be written as an infinite sequence of substitutions,

$$e =< s_i | \quad s_i \in S \quad 0 \leq i < \infty >$$

Let E be the set of all enciphering functions from Σ_1^+ to Σ_2^+.

Definition 3.3 *A substitution cryptographic system, c, is a function*

$$c : \Sigma_1^+ \times K \to \Sigma_2^+$$

such that, for each $k \in K, c(\cdot, k) \in E$.

$c(\cdot, k)$ is an infinite sequence of substitutions, $< s_i(\cdot, k) |\ s_i(\cdot, k) \in S,\ \ 0 \leq i < \infty >$.

$c(\cdot, k)$ is the *enciphering function* and its inverse is the *deciphering function.* i.e

$$c(x, k) = y \ \text{ and } \ x = c^{-1}(y, k)$$

Applying c to a plaintext string, $x = x_0 x_1 \cdots x_{n-1}$, gives a ciphertext string, $y = y_0 y_1 \cdots y_{n-1}$, where

$$\begin{aligned} c(x, k) &= s_0(x_0, k) s_1(x_1, k) \cdots s_{n-1}(x_{n-1}, k) \\ &= y_0 y_1 \cdots y_{n-1} \\ &= y. \end{aligned}$$

Most cryptosystems are substitution systems, i.e. they map each plaintext letter uniquely to a ciphertext letter. This is generally a desirable property, since if a ciphertext string can decrypt to numerous plaintext strings the intended receiver needs some way of destinguishing between the possible decryptions. One-way hash functions are an example of the use of a many-to-one function in cryptography [14]. One-to-many functions are used in *probabilistic encryption* [30, 29]. We concentrate on substitution systems. We assume $q_1 = q_2 = q$, and $\Sigma_q = Z_q$, unless otherwise stated, in which case a substitution is a permutation on Z_q. The set of all substitutions on Z_q is

the *symmetric group* of degree q, denoted S_q.

Definition 3.4 *A substitution cryptographic system, c, is called monoalphabetic if,* $\forall k \in K$*, the infinite sequence of substitutions of the encryption function,*

$$c(\cdot, k) =< s_i(\cdot, k) | \;\; s_i(\cdot, k) \in S \; 0 \leq i < \infty >,$$

are all identical, i.e. if

$$s_i(\cdot, k) = s_j(\cdot, k), \;\; 0 \leq i, j < \infty.$$

Otherwise c is called a *polyalphabetic substitution system.*

Thus for any key, k, a monoalphabetic enciphering function, $c(\cdot, k)$, is fully described by the first substitution in the sequence

$$c(\cdot, k) = s_0(\cdot, k)s_1(\cdot, k) \cdots.$$

Definition 3.5 *A substitution system, c, is* periodic*, with period P, if,* $\forall k \in K$*, the substitutions in the sequence*

$$c(\cdot, k) =< s_i(\cdot, k) | \;\; s_i(\cdot, k) \in S \; 0 \leq i < \infty >,$$

satisfy

$$s_i(\cdot, k) = s_{i+P}(\cdot, k), \;\; 0 \leq i < \infty.$$

For any key, k, a periodic polyalphabetic substitution enciphering function, $c(\cdot, k)$, with period P, is then fully described by the first P substitutions in the sequence of substitutions

$$c(\cdot, k) = s_0(\cdot, k)s_1(\cdot, k) \cdots s_{P-1}(\cdot, k) \cdots.$$

3.2 Cryptanalysis

This section briefly looks at the possible problems facing the cryptanalyst and how a cryptosystem can be designed to make the cryptanalysts task as hard as possible. Most substitution systems try and achieve security by

- using a large keyspace and
- using a large alphabet or a large number of substitutions.

The problem with increasing the alphabet size or using a large period is the specification of the particular substitutions required for any key. These substitutions have to be generated by some function over the keyspace, as listing them explicitly will be too large a task. This introduces another area of possible weakness, for if the cryptanalyst can work back from the ciphertext and plaintext to the key, he can break the system. Cryptanalysis is the study of recovering the plaintext from ciphertext without knowledge of the key or alternatively recovering the key from plaintext and ciphertext. A good cryptosystem will ensure that neither is practicable without the key even when the cryptosystem is known. It is generally assumed that the cryptosystem, c, is known. The three main modes of attack are as follows.

- Ciphertext only attack. In this case the cryptanalyst knows only a ciphertext string, y, encrypted with a key, k, and the cryptosystem, c, or possibly ciphertext strings $y_1, y_2, \ldots, y_j$ encrypted with unknown keys $k_1, k_2, \ldots, k_j$. The aim of cryptanalysis is to recover the plaintext, x, the key, k, or some way of deciphering other messages enciphered with k without necessarily recovering the key, i.e. find an algorithm to infer $x' = c^{-1}(y', k)$ where $x' \in \Sigma_1^+,\ y' \in \Sigma_2^+$, given y and c.

- Known plaintext attack. The cryptanalyst knows some ciphertext string, y and the corresponding plaintext string, x, and c. A successful attack will identify k or an algorithm to infer $x' = c^{-1}(y', k)$ where $x' \in \Sigma_1^+,\ y' \in \Sigma_2^+$, given x, y and c.

- Chosen plaintext attack. The cryptanalyst is able to obtain the ciphertext string, y, for some chosen plaintext string, x. A successful attack will identify k or an algorithm to infer $x' = c^{-1}(y', k)$ where $x' \in \Sigma_1^+,\ y' \in \Sigma_2^+$, given x, y and c.

Cryptanalysis with ciphertext only is possible because of the *redundancy* of a language.

If we define a random variable, X_i, with range Z_q, and probability distribution

$$p(t) = P(X_i = t) \quad 0 \leq t < q,$$

then the entropy of X_i is defined as

$$H(X_i) = -\sum_{t=0}^{q-1} p(t) \log_2 p(t).$$

(If $p(t) = 0, p(t) \log p(t)$ is evaluated as 0.) Let plaintext message, x, be an observation of n identically distributed random variables, $X_0, X_1, \ldots, X_{n-1}$, then the entropy of $X_0, X_1, \ldots, X_{n-1}$ is

$$H_n = H(X_0, X_1, \ldots, X_{n-1}) = -n \sum_{t=0}^{q-1} p(t) \log_2 p(t).$$

The average entropy per letter of a language, also called the *rate* of a language, r, is the limiting value of

$$r = \lim_{n \to \infty} \frac{H_n}{n},$$

where r is measured in bits per letter. The absolute rate for a language is the average entropy per letter assuming each is equally likely, i.e.

$$R = \log_2 q.$$

The redundancy of a language, D, is then

$$D = R - r.$$

So, if we assume that plaintext originates from independent observations of a random variable with single letter probabilities given in Appendix A,

$$r = 4.127 \quad \text{and} \quad R = 4.7$$

so

$$D = 0.573.$$

Measuring the redundancy in actual English is more complicated, because the letters are not independent. The rate of normal English has been estimated as between 1.0 and 1.5 (Shannons estimated a rate of 1.2 [72] using a technique called randomisation). The redundancy of English then is approximately 3.5 bits/letter.

The difference between the single letter model and actual English illustrates the point that a model based on single letter frequencies is not particularly good. English provides about three bits per letter extra information than the single letter model.

3.2.1 Unconditional and Computational Security

A cryptosystem is *unconditionally secure*, or *theoretically secure*, if it is secure against an attacker with unlimited computational resources. It is *computa-*

tionally secure, or *practically secure*, if it is secure against an opponent with a specified limited computational power.

Unconditional Security

Definition 3.6 *A cryptographic system provides perfect secrecy if the ciphertext letters and the plaintext letters are statistically independent.*

Suppose we consider n random variables, $X_0, X_1, \ldots, X_{n-1}$, defined on the plaintext alphabet and $Y_0, Y_1, \ldots, Y_{n-1}$, defined on the ciphertext alphabet. $X_0, X_1, \ldots, X_{n-1}$ and $Y_0, Y_1, \ldots, Y_{n-1}$ are independent iff

$$P(X_i = x_i \cup Y_j = y_j) = P(X_i = x_i)P(Y_j = y_j) \;\; 0 \leq i, j, < n.$$

Shannon showed that perfect secrecy is obtainable, although it requires a key space as large as the set of all possible messages. Such a system is the *one time pad*, [70], which is a substitution system with a number of substitutions equal to the number of letters in the message (usually defined on a binary alphabet). As long as the key (in this case the set of substitutions) remains unknown to the attacker and is not reused, the system is impossible to break. Perfect secrecy is impractical for all but the highest priority, infrequent communications. For cryptographic systems with less than perfect secrecy (i.e. for all cryptosystems except variants of the one time pad) there is some dependency between ciphertext and plaintext. This dependency is what the cryptanalyst hopes to exploit to recover the plaintext. But this dependency is only useful if there is some redundancy in the language. Unconditional security is also achievable if a language has no redundancy. Shannon defined a cryptographic system to be *strongly ideal* if, for a given key, the ciphertext

gives no information about the key. If the plaintext letters are independent and equally likely then almost any substitution system will be strongly ideal. A strongly ideal system is secure against a ciphertext only attack. The redundancy in a language can be reduced by data compression, but cannot be entirely removed. Although mainly of theoretical interest, the ideas behind the unconditionally secure systems indicate good principles for the utilisation of a cryptosystem, namely that

- the secret key should be changed as frequently as possible
- the plaintext letters should be as random as possible.

Computational Security

The *complexity* of an attack means the average number of operations used in that attack. A cryptosystem is computationally secure if the complexity of the *optimum* attack exceeds the computational capability of the cryptanalyst. Generally it is not possible to determine whether an attack is optimal. Instead we use the best presently known method of attack to evaluate the security. For any attack the *data complexity* is the average amount of input data in the form of ciphertext or plaintext/ciphertext pairs required, and the *processing complexity* is the average amount of computations needed to process the data to cryptanalyse a cryptosystem. The dominant quantity is called the complexity for the attack.

3.3 Rotor Machines

Rotor machines formed the basis for most military and commercial cryptography until the late 1950s. The German Enigma, the Hebern machine and the Converter M-325, all described in [17], are variations on the basic machine we use.

Rotor machines utilise Caesar substitutions.

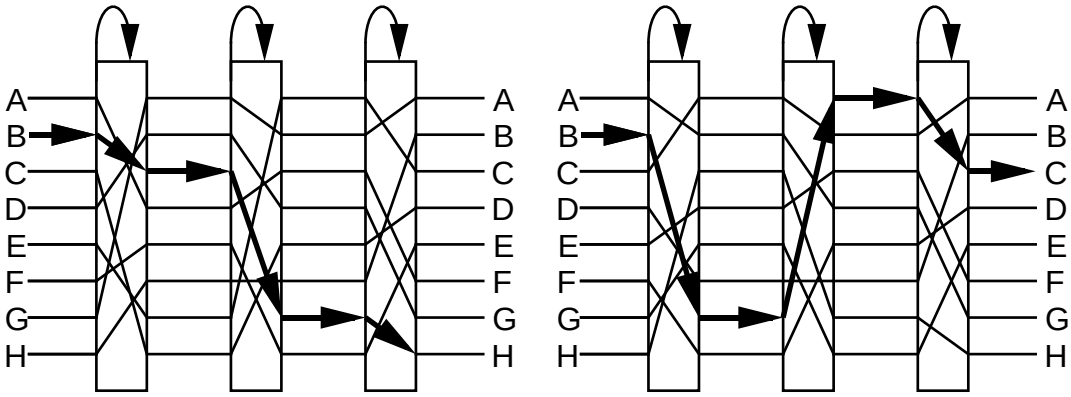

Figure 3.3: A three rotor machine for an eight letter alphabet before and after the first rotor has rotated one place.

Caesar Systems

The subgroup $\mathbf{C}_q = \{C_s : 0 \leq s < q\}$ of S_q containing the q substitutions

$$C_s : j \to (j + s) \quad 0 \leq s,\ j < q,$$

is the group of *Caesar substitutions*.

A *Caesar system* is a monoalphabetic substitution that enciphers x into y according to the rule

$$y_i = C_s(x_i) \quad 0 \leq i < n.$$

A rotor machine is a cryptographic device consisting of N rotors. A rotor is a disk with a node on each side for each letter in the appropriate alphabet (size q) and electrical contacts between the nodes effecting a substitution π. After each such substitution the contacts rotate in accordance with a rotation pattern while the nodes remain in the same position (or vice versa). Rotating a rotor a places yields another (usually different) substitution,

$$S(a, \pi) = C^{-a} \pi C^{a}.$$

$S(a, \pi)$ is the composition of π with the Caesar substitutions C^{-a} and C^{a}. Rotor machines are built by concatenating rotors. Let

$$\Pi = \{\pi_0, \pi_1, \ldots, \pi_{m-1}\}$$

be a set of substitutions representing a bank of m rotors for encipherment of the first letter and

$$R = \{r_0, r_1, \ldots, r_{m-1}\}$$

be a set of *displacement functions*,

$$\begin{aligned} &r_s : Z_q \to Z_q \\ &r_s : 0 \to 0 \\ &r_s : i \to r_s(i), \quad 0 \leq s < m, \quad i > 0 \end{aligned}$$

where $r_s(i)$ is the rotational displacement of the s^{th} rotor for the encipherment of the i^{th} element of $\mathbf{x}$. The enciphering substitution for the i^{th} letter of plaintext is given by

$$S(R(i), \Pi) = C^{-r_{m-1}(i)} \pi_{m-1} C^{r_{m-1}(i)} C^{-r_{m-2}(i)} \pi_{m-2} C^{r_{m-2}(i)} \cdots C^{-r_0(i)} \pi_0 C^{r_0(i)}$$

and the deciphering substitution is the inverse,

$$S(R(i),\Pi)^{-1} = C^{-r_0(i)}\pi_0^{-1}C^{r_0(i)}C^{-r_1(i)}\pi_1^{-1}C^{r_1(i)}\cdots C^{-r_{m-1}(i)}\pi_{m-1}^{-1}C^{r_{m-1}(i)}.$$

The set of displacement functions is usually assumed to be known. A common set of displacement functions are those that follow an odometer like rotation pattern, such as

$$r_s(i) = \lfloor (i/q^s) \rfloor \;\; 0 \leq s < m, \;\; 0 < i < \infty.$$

The key for a rotor machine must define:

1. the number of rotors, m,
2. the initial rotor substitutions, Π,
3. the displacement functions, R.

We assume the number of rotors is known and that the set of displacement functions follow the odometer like pattern described above. So the keyspace, K, is the set of all m-tuples of possible substitutions and a key, k, is an element of a subset of K.

An m rotor cryptosystem with odometer like displacement functions, R, is a *polyalphabetic substitution system*, $c : \Sigma^+ \times K \rightarrow \Sigma^+$, defined on alphabet $\Sigma = Z_q$, where, $\forall k \in K$,

$$c(\cdot, k) =< S(R(i), k) \;\; | \;\; 0 \leq i < \infty >,$$

with period P, which divides q^m. A rotor machine with a set of displacement functions which are not odometer like can, at most, produce q^m different substitutions. A fuller mathematical description of rotor machines is given in [45].

3.4 Block Cipher Systems

Block cipher systems are simple substitution systems which achieve security through increasing the size of the alphabet and employing a large keyspace. [48] gives a thorough description of block ciphers. Suppose we consider an alphabet which consists of all possible bit strings of length t. Our alphabet is then $\Sigma = Z_{2^t}$ and a letter is

$$x_i = a_0 a_1 \ldots a_{t-1}. \text{ where } a_j = 0 \text{ or } 1.$$

A letter can also be considered as a binary representation of an integer

$$x = \sum_{i=0}^{t-1} x_i 2^{t-1-i}.$$

A block cryptosystem is a simple substitution system on alphabet Z_{2^t}.
A key, k, is generally a binary string, the length (number of bits) of which is denoted r. The keyspace, K, is Z_{2^r}. If the key for a block cipher simply randomly specified one of the $2^t!$ substitutions on Z_{2^t}, an exhaustive search of the keyspace is the only option for the cryptanalyst. The keyspace, then, must be large enough to make an exhaustive search of the keyspace infeasible. The key is normally at least 56 bits long. But specifying a substitution on Z_{2^t} explicitly requires the listing of 2^t mappings, which is infeasible for any t of reasonable size (t is generally at least 64). Instead, block ciphers generate a function, B,

$$B : \Sigma \times K \to \Sigma,$$

in such a way that the relationship between the plaintext, ciphertext and key is complex enough to make cryptanalysis infeasible. i.e. the inevitable dependence between plaintext and ciphertext is complex enough to reveal

little about the key. If, as is usual, B, is found by the repeated application of simpler functions, the resulting block cryptosystem is called an *iterated cipher*.

Iterated Block Ciphers

Each iteration of an iterated cipher is called a round. The function applied to a letter at each round is called the round function, f,

$$f : \Sigma \times K \rightarrow \Sigma,$$

where the key for the j^{th} round, k_j, is derived from the full secret key , k, by a *key scheduling algorithm* (see figure 3.4). The round function for the j^{th} round is denoted f_j.

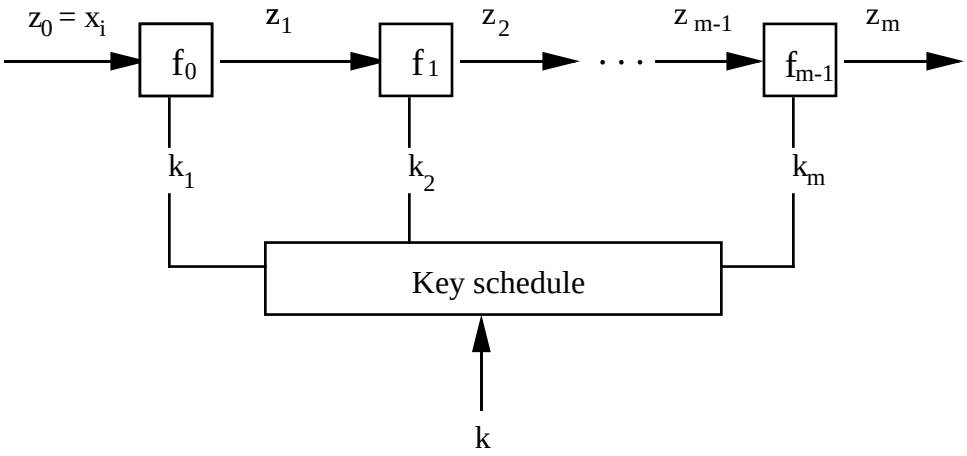

Figure 3.4: An iterated block cipher with m-rounds

Iterated block ciphers are designed so that encipherment and decipherment can be achieved using the same key. The round function is derived

from the composition of up to three types of function.

Involutionary Permutation. A function, P, on Σ, is an involution, or an involutionary permutation, if $P(P(x)) = x$ for all $x \in \Sigma$.

For example, any composition of independent transpositions on Σ is an involution. Similarly, if a letter, x, is considered as a bit string, swapping the left and right bit blocks of x is also an involutionary permutation.

Key Dependent Involution Function. A function $I(\cdot, \cdot)$,

$$I : \Sigma \times K \to \Sigma,$$

is an involution cipher if, for every choice of key, k, $I(\cdot, k)$ is an involution on Σ.

Group Function. A binary function, $G(\cdot, \cdot)$,

$$G : \Sigma \times \Sigma \to \Sigma$$

is a group function if (Σ, G) is a group. Note that for a group function the key space must equal the letter space.

Involution Cryptosystems

If the round function, f, involves a key dependent involution followed by an involutionary permutation,

$$f_j = f(x_i, k_j) = P(I(x_i, k_j)),$$

the resulting function

$$B : \Sigma \times K \to \Sigma$$

$$B = f_r \circ f_{r-1} \circ \cdots \circ f_1$$

forms an involution cryptosystem. DES [58], FEAL [73] and LOKI [10] are all involution cryptosystems.

Group and Involution Systems

If the round function, f, involves a group function followed by an involutionary permutation,

$$f_j = f(x_i, k_j) = I(G(x_i, k_j^A), k_j^B),$$

the resulting function

$$B : \Sigma \times K \to \Sigma$$

$$B = f_r \circ f_{r-1} \circ \cdots \circ f_1$$

forms a group and involution system. Note that they use two key schedules, one for the group function and one for the involution function. PES [49] is a group and involution system.

Group, Involution and Involutionary Permutation Systems

If the round function, f, involves a key dependent involution and a group function followed by an involutionary permutation,

$$f_j = f(x_i, k_j) = P(I(G(x_i, k_j^A), k_j^B))$$

The resulting function

$$B : \Sigma \times K \to \Sigma$$

$$B = f_r \circ f_{r-1} \circ \cdots \circ f_1$$

forms a group, involution and involutionary permutation system, an example of which is IDEA [48].

3.4.1 Data Encryption Standard (DES)

DES is an involution cryptosystem on $Z_{2^{64}}$. It uses a 56 bit key and 16 iterations of an involution function followed by an involutionary permutation.

Key Scheduling Algorithm

The key bits used on the i^{th} round of DES are defined inductively. The user supplied 56 bit key is initially permuted. The resulting 56 bit vector is split into L_0 and R_0, the left and right 28 bit blocks. Let λ^i be a left circular shift of i places, and σ be any array specifying the number of shifts at each round (either one or two shifts). Then at each round

$$L_i = \lambda^{\sigma[i]} L_{i-1} \quad \text{and} \quad R_i = \lambda^{\sigma[i]} R_{i-1}.$$

The 48 bit key for the i^{th} round, k_i, is found by means of a *compression permutation* on $L_i R_i$.

Involutionary Permutation

The involutionary permutation, V, is a swapping of the two 32 bit halves of a 64 bit string, x.

Key Dependent Involution

Let Σ_2 be the alphabet of all 32-bit letters. if we define the function, π, as

$$\pi : \Sigma_2 \times \Sigma_2 \to \Sigma_2 \times \Sigma_2$$

$$\pi(x_l, x_r) = (x_l + T(x_r, k), x_r),$$

where addition is a bitwise operation modulo 2, and

$$T : \Sigma_2 \times K \to \Sigma_2,$$

then

$$\begin{aligned} \pi^2(x_l, x_r) &= \pi(x_l + T(x_r, k), x_r) \\ &= (x_l + T(x_r, k) + T(x_r, k), x_r) \\ &= (x_l, x_r). \end{aligned}$$

So π is an involution function. The following paragraph describes the operation of π on each round.

Let $x^{(i)}$ be the 64 bit input on the i^{th} round, then $x^{(i)}$ is split in half into two 32 bit blocks, and the right 32-block, $x_r^{(i)}$, is expanded into 48 bits by an *expansion permutation*. If the eight 4-bit blocks of $x_r^{(i)}$ be are written as an

(8×4) matrix,

$$\begin{matrix} x_0^i & x_1^i & x_2^i & x_3^i \\ x_4^i & x_5^i & x_6^i & x_7^i \\ x_8^i & x_9^i & x_{10}^i & x_{11}^i \\ x_{12}^i & x_{13}^i & x_{14}^i & x_{15}^i \\ x_{16}^i & x_{17}^i & x_{18}^i & x_{19}^i \\ x_{20}^i & x_{21}^i & x_{22}^i & x_{23}^i \\ x_{24}^i & x_{25}^i & x_{26}^i & x_{27}^i \\ x_{28}^i & x_{29}^i & x_{30}^i & x_{31}^i \end{matrix}$$

the eight 6-bit blocks are found by copying the outer bit positions from the previous and next rows into the first and last position respectively, giving

$$\begin{matrix} x_{31}^i & x_0^i & x_1^i & x_2^i & x_3^i & x_4^i \\ x_3^i & x_4^i & x_5^i & x_6^i & x_7^i & x_8^i \\ x_7^i & x_8^i & x_9^i & x_{10}^i & x_{11}^i & x_{12}^i \\ x_{11}^i & x_{12}^i & x_{13}^i & x_{14}^i & x_{15}^i & x_{16}^i \\ x_{15}^i & x_{16}^i & x_{17}^i & x_{18}^i & x_{19}^i & x_{20}^i \\ x_{19}^i & x_{20}^i & x_{21}^i & x_{22}^i & x_{23}^i & x_{24}^i \\ x_{23}^i & x_{24}^i & x_{25}^i & x_{26}^i & x_{27}^i & x_{28}^i \\ x_{27}^i & x_{28}^i & x_{29}^i & x_{30}^i & x_{31}^i & x_0^i \end{matrix}$$

The expanded text is then bitwise added to the key for the i^{th} round, k_i, and the result is split into 8 blocks of 6 bits. These blocks form the input to the *substitution boxes* (S-boxes), S, each of which has a 4-bit output. Each box, $S[j]$, consists of a (4×16) array, each row of which specifies a permutation on Z_{16}. The first and last bit of input specify which row of $S[j]$ to use, and the middle 4 bits are then permuted by that row. The 32 bit output

of the s-boxes is permuted once more, and then bitwise added to the left 32-block input, x_l^i.

Round Function

Let s be the splitting function, i.e. a function which splits a 64 bit block into right and left 32 bit blocks,

$$s : \Sigma \to \Sigma_2 \times \Sigma_2,$$

then the round function for the i^{th} round,f_i, using key k_i, is

$$f_i = s^{-1} \circ V \circ \pi_i \circ s$$

The DES encryption function is

$$y = DES(x) = P^{-1} \circ f_{16} \circ \cdots f_2 \circ f_1 \circ P(x)$$

where P is a permutation on Z_{64} constant for all keys. The decryption function is

$$x = DES^{-1}(y) = P^{-1} \circ f_1 \circ \cdots f_{15} \circ f_{16} \circ P(y).$$

3.5 Stream Cipher Systems

Stream cipher systems are periodic polyalphabetic substitution systems which use the binary alphabet (Z_2) for plaintext and ciphertext, but achieve security through having a large keyspace and a large period. A *keystream generator* has the key as an input and generates a stream of bits, each of which is added

modulo 2 (XORed) with the appropriate plaintext letter. The key must obviously change for each message to foil a known plaintext/ciphertext attack, since XORing the plaintext with the ciphertext will give the keystream, which will be identical if the same key is reused for another message. The generator must produce a bit stream which has a very large period to counter a statistical ciphertext only attack. If the generator produced a truly random continual stream of bits it would be a one-time pad, but as already said this requires a key as long as the message. Stream ciphers, then, rely wholly on the keystream generator for their security. Synchronous stream ciphers generate the keystream independent of the message by means of pseudo random number generators (PRNGs), several of which are described in [67]. Self-synchronous stream ciphers generate the keystream bits as a function of previous ciphertext bits. Linear congruental generators [43, 44, 50], linear feedback shift registers [31, 51] and feedback with carry shift registers [42] are all self-synchronous stream ciphers.

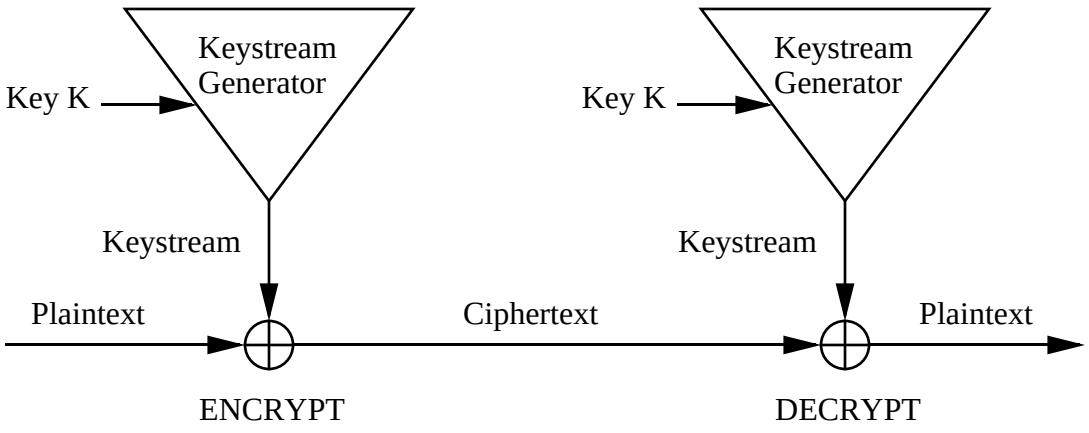

Figure 3.5: Model of a stream cipher

3.6 Public Key Systems

A prerequisite of secret key systems is that the key can be sent to all intended receivers securely. This is an expensive and dangerous process and severely limits the scope of a cryptographic system. Public key cryptography, first described in [20], overcomes this problem by using a public/private pair of keys. The idea is that the sender uses the receivers public key, obtainable by everyone, to encrypt the message, and the receiver uses his private key, known only to him, to decrypt. The relationship between the public and private key is such that finding the private key from the public key is impractical. Most public key systems are simple substitution systems defined over a large alphabet and keyspace, i.e. they are block ciphers with the important difference that they do not use involutions, and the inverse of the encryption function is not computable without the secret part of the key. This is possible by the use of trap door functions. Most public key systems also use a different alphabet for plaintext and ciphertext.

Trap Door and One Way Functions

A function, f, is *easy to compute* if there exists an algorithm to compute f in polynomial time. A function, f, is *hard to compute* if any algorithm to compute f is not known to operate in polynomial time.

A function, f, is a *one way function* if f is easy to compute and f^{-1} is hard to compute.

Definition 3.7 *A function, f, is a trap door function if f is easy to compute and f^{-1} is hard to compute unless some information (a private key) is known, in which case f is easy to compute.*

So, a public key cryptosystem is a function, P, with a keyspace, K, of pairs of key, public and private,

$$P : \Sigma \times K \to \Sigma$$

where $P(\cdot, k)$ is a trap door function, $\forall k \in K$.

3.6.1 Knapsack Public Key Algorithms

The Knapsack Problem

Given a vector of n values, $\boldsymbol{a} = (a_0, a_1, \ldots, a_{n-1})$, and a sum, S, find $x = (x_0, x_1, \ldots, x_{n-1})$ such that

$$S = \sum_{i=0}^{n-1} x_i a_i,$$

where $x_i = 0$ or 1, for $0 \leq i < n$.

The knapsack problem is NP-hard, but certain types of knapsack problem are easy, in the sense that all instances of this type are solvable in linear time. Knapsack public key algorithms utilise this fact by transforming an easy knapsack problem into a hard one. A knapsack algorithm must specify what class of easy knapsack problem is to be used and how it is to be transformed into a hard knapsack problem.
The general class of knapsack systems can be summarised as follows.

- Public key: A vector of values, $\boldsymbol{a} = (a_0, a_1, \ldots, a_{n-1})$.
- Private key: A method of transforming a knapsack problem with values $\boldsymbol{a}$ into an easy knapsack problem.
- Plaintext: $x = (x_0, x_1, \ldots, x_{n-1})$ where $x_i = 0$ or 1.
- Encryption: Ciphertext y is an integer such that $y = x\boldsymbol{a'}$.
- Decryption: Solve knapsack problem with weights $\mathbf{a}$ and sum y.

Merkle-Hellman Knapsack Algorithm

The Merkle-Hellman cryptosystem takes a super-increasing sequence of numbers as the weights for an easy knapsack problem and uses modular multiplication to derive a hard knapsack problem.

Super-Increasing Knapsack Problem

A super-increasing sequence of numbers is one where every element is greater than the sum of all the previous elements in the sequence.

$$s_j > \sum_{i=0}^{j-1} s_i \quad 0 \leq j < n.$$

A super-increasing knapsack problem is one that uses a super-increasing vector of weights. They can be solved using the simple algorithm

for $i = n - 1$ down to 0 **do**

if $s_i \leq S$

$x_i = 1$ and $S = S - s_i$;

else

x_i=0;

end **do**;

To transform the super-increasing sequence into a hard knapsack problem, choose two numbers M and W such that

$$M > \sum_{i=0}^{n-1} s_i \text{ and } \gcd(W, M) = 1.$$

The public key vector of weights is then $\boldsymbol{a}$, where $a_i = s_i W \pmod{M}$.

The private key is the vector of weights $\boldsymbol{s}$ and the two numbers M and W. With the private key it is easy to recover the plaintext. Let

$$S = yW^{-1} \pmod{M}$$

then solving the knapsack problem with weights $\boldsymbol{s}$ and sum S recovers the plaintext $\mathbf{x}$.

There are many alternative knapsack type public key cryptosystems, a review of which is given in [19]. The basic Merkle-Hellman cryptosystem was cracked by Shamir [71], and all proposed variants except [12], which is based on arithmetic in a finite field, have been cracked using techniques described in [9].

3.6.2 RSA

Named after its inventors, Rivest, Shamir and Adleman [64], it is the most popular public key system. The basis of RSA is the difficulty in factoring very large numbers. A public key, (n, e), is generated by firstly finding the product of two large (over 100 digits) prime numbers, $n = p \times q$, and secondly choosing a random e such that e and $\phi(n)$ are coprime, where $\phi(n) = (p-1)(q-1)$ is *Euler's totient function*. A private key, d, which satisfies

$$e \times d = 1 \pmod{\phi(n)}$$

is found using Euclid's algorithm.

The plaintext alphabet, Σ_1, is of a magnitude less than n. A simple substitution on Σ_1 is generated by exponentiation to the power e modulo n. i.e. ciphertext letters, y_i, are generated from plaintext letters, x_i,

$$y_i = x_i^e \pmod{n}.$$

The inverse substitution on Σ_2 is generated by exponentiation to the power d modulo n. i.e plaintext letters are recovered from ciphertext letters by

$$x_i = y_i^d \pmod{n}.$$

It is conjectured that the security of RSA depends on the difficulty of factoring large numbers.

Other public key systems include ElGalmal [21], Lin Chan Lee [52] and Goldwasser-Micali [30].

Chapter 4

Cryptanalysis: A Statistical Approach

4.1 Introduction

This chapter looks at how the ideas of statistical inference are applied in cryptanalysis. The classical role of statistics in cryptanalysis is to utilise the inherent non-randomness of a language to attack cryptosystems with ciphertext only. Modern cryptosystems have made this approach almost impossible by using very large alphabets or periods. Statistics is still useful to test the strength of systems by measuring the dependence between ciphertext and plaintext and is central to the technique of *differential cryptanalysis*, which looks at the statistical differences of pairs of plaintext and ciphertext. The second section uses the simplest cryptographic systems as a means to illustrate some of the most basic cryptographic techniques used in ciphertext only

attacks. Section three describes a statistical technique for the cryptanalysis of rotor machines based on the maximum likelihood estimator. The fourth section describes some of the applications of statistics in highlighting possible weaknesses in block ciphers, and the final section looks at cryptanalysis and public key systems.

4.2 Cryptanalysis of Basic Systems

Cryptographic substitution systems with a small alphabet and period are susceptible to a ciphertext only attack using the statistical properties of the plaintext language if that language has redundancy, i.e. if the plaintext letter distribution is not uniform.

4.2.1 Monoalphabetic Substitution Systems

Cryptanalysis of The Caesar System

The inferential problem is to identify the correct key, and hence the correct plaintext, from the sample data, the ciphertext, y, assuming we know the cryptosystem and that the plaintext is derived from independent random sampling of the single letter frequency distribution given in Appendix A. The parameter we wish to estimate is the key, k, defined on the keyspace $K = Z_{26}$.

We may choose to use the *maximum likelihood estimate* of k or, from a Bayesian viewpoint, choose the value of k that maximises our posterior distribution. The ML estimate of k is the value of k that maximises

$$L(y|k) = P(y_0|k)P(y_1|k)\cdots P(y_{n-1}|k).$$

The Bayesian point estimate of k is the value of k that maximises the posterior distribution

$$\pi(y|k) = \frac{\pi(k)L(y|k)}{\sum_K \pi(k)L(y|k)}.$$

For example, suppose the message x=SENDHELP is enciphered using the Caesar system, with key $k = 7$, so that the ciphertext is y=ZLUKOLSW. Suppose also that we have no prior knowledge of the possible key value, so our prior estimate of the key values is that they are all equally likely.

$$\pi_1(k) = \frac{1}{26}, \quad 0 \leq k < 26.$$

Alternatively, suppose that we do have some prior knowledge. Previous successful attacks show that the keys $k = 4$ to $k = 9$ are twice as likely to be used as the other keys. In this case our alternative prior, π_2, is

$$\pi_2(k) = \begin{cases} 2/32 & \text{for } \ 4 \leq k \leq 9, \\ 1/32 & \text{otherwise,} \end{cases}$$

The second column of table 4.1 shows that the ML estimate of k is $\hat{k} = 18$. The third column of table 4.1 indicated that the Bayesian point estimate, with prior knowledge, is $\hat{k} = 7$. The reason that the maximum likelihood estimate gives the wrong solution is the simplicity of the 1-gram model and the small amount of ciphertext available. With the benefit of prior information we were able to correctly decipher the message, but it must be emphasised that if a key other than $k = 4$ to $k = 9$ had been used we would have been less likely to identify the correct plaintext. One of the reasons the Caesar substitution offers no security is that the keyspace is small enough to try every key. However, a large keyspace does not guarantee security.

k	(y - k) mod m	$\pi_1(y\|k)$	$\pi_2(y\|k)$
0	ZLUKOLSW	0.000055	0.000040
1	YKTJNKRV	0.000005	0.000004
2	XJSIMJQU	0.000000	0.000000
3	WIRHLIPT	0.051953	0.037891
4	VHQGKHOS	0.000034	0.000050
5	UGPFJGNR	0.000088	0.000129
6	TFOEIFMQ	0.005256	0.007666
7	**SENDHELP**	0.364221	**0.531275**
8	RDMCGDKO	0.001518	0.002215
9	QCLBFCJN	0.000003	0.000004
10	PBKAEBIM	0.000865	0.000631
11	OAJZDAHL	0.000294	0.000215
12	NZIYCZGK	0.000003	0.000002
13	MYHXBYFJ	0.000001	0.000001
14	LXGWAXEI	0.000093	0.000068
15	KWFVZWDH	0.000003	0.000002
16	JVEUYVCG	0.000010	0.000007
17	IUDTXUBF	0.000397	0.000289
18	**HTCSWTAE**	**0.567588**	0.413960
19	GSBRVSZD	0.000155	0.000113
20	FRAQURYC	0.000563	0.000411
21	EQZPTQXB	0.000000	0.000000
22	DPYOSPWA	0.006490	0.004733
23	COXNROVZ	0.000113	0.000083
24	BNWMQNUY	0.000040	0.000029
25	AMVLPMTX	0.000251	0.000183

Table 4.1: Posterior distributions of the key for a Caesar cipher

The keyspace for a monoalphabetic substitution system defined on Z_{26} has size 26!. Despite the size of this keyspace, these systems offer very little security. The reason for this is the statistical properties of the plaintext language, such as the letter E being much more common than the letter Z, are also apparent in the ciphertext. The letter substituting E will appear as often in the ciphertext as E does in the plaintext. Several methods of cracking monoalphabetic substitutions have been suggested [37, 2, 61, 11], but it is generally not a hard problem for the cryptanalyst, and a simple heuristic search technique will usually suffice. For this reason it is often of interest to know whether a particular ciphertext string has been enciphered monoalphabetically. To test whether a string of text of size n was enciphered by a monoalphabetic substitution we use the ϕ *statistic*, defined to be

$$\phi = \sum_{i=0}^{q-1} f_i(f_i - 1),$$

where f_i is the observed frequency of the i^{th} letter. then

$$\mu_\phi = E(\phi) = s_2 n(n-1), \tag{4.1}$$

and the variance is

$$\sigma_\phi^2 = 4n^3(s_3 - s_2^2) + 2n^2(5s_2^2 + s_2 - 6s_3) + 2n(4s_3 - s_2 - 3s_2^2), \tag{4.2}$$

where

$$s_2 = \sum_{i=0}^{q-1}(P(i))^2$$

and

$$s_3 = \sum_{i=0}^{q-1}(P(i))^3.$$

All the results in the section are given in more detail and proved in [46], which also provides tables for each statistic for different values of n. For random text, where

$$P(i) = \frac{1}{q} \text{ for } 0 \leq i < q,$$
$$E(\phi_r) = 0.038n(n-1)$$

and

$$\sigma^2_{\phi_r} = 0.073964n(n-1).$$

For text arising from observation of the distribution given in Appendix A,

$$E(\phi_m) = 0.0661n(n-1),$$

and

$$\sigma^2_{\phi_m} = 0.004344n^3 + 0.110448n^2 - 0.114794n.$$

If it is assumed that ϕ is normally distributed, with mean μ_ϕ and variance σ^2_ϕ, the probability of observing a text string with a ϕ value in range $[A, B]$ arising from any known distribution, is given by

$$P(A < \phi < B) = P\left(\frac{A-\mu}{\sigma} < Z < \frac{B-\mu}{\sigma}\right),$$

where Z is a random variable following the standard normal distribution ($Z \sim N(0,1)$). a hypothesis test can then easily be constructed to test

$$H_0 : \mu_\phi = \mu_{\phi_r}, \sigma_\phi = \sigma_{\phi_r}$$
$$H_1 : \mu_\phi = \mu_{\phi_m}, \sigma_\phi = \sigma_{\phi_m}$$

Matching Alphabets

It is often of interest to determine whether two ciphertext strings which are known to be monoalphabetically enciphered have been enciphered with the same substitution. Given two strings of ciphertext, y_1 and y_2, of lengths n_1 and n_2, we are interested in testing

H_0 : y_1 and y_2 are enciphered by two different monoalphabetic substitution systems, S_1, S_2.

H_1 : y_1 and y_2 are enciphered by the same monoalphabetic substitution system, S.

Application of the ϕ Test

If two non random distributions with letter frequencies $f'_0+f'_1+\ldots+f'_{q-1} = n_1$ and $f''_0 + f''_1 + \ldots + f''_{q-1} = n_2$ are combined so that $f_0 = f'_0 + f''_0,\ f_1 = f'_1 + f''_1,\ \ldots,\ f_{q-1} = f'_{q-1} + f''_{q-1},\ n = n_1 + n_2$, then the expected value of ϕ is as given by 4.1 and the variance by 4.2. If the alphabets match (i.e. both strings are from the same distribution) then, using the single letter frequencies, the mean and variance are $E(\phi_m)$ and σ_{ϕ_m}. If the alphabets do not match 4.1 and 4.2 become

$$E(\phi) = 0.066n(n-1) - 0.056n_1n_2$$

$$\begin{aligned}\sigma^2_\phi &= 0.004344(n_1^3 + n_2^3) + 0.110448(n_1^2 + n_2^2) - 0.114794(n_1 + n_2) \\ &\quad +4n_1n_2[(n_1 + n_2)(0.001063) + 0.034856].\end{aligned}$$

For a full derivation of the ϕ and χ statistics for non-matching alphabets see Appendix E of [46]

Application of the χ Test

If we define the statistic χ as

$$\chi = f'_0 f''_0 + f'_1 f''_1 + \ldots f'_{q-1} f''_{q-1},$$

then, if the two alphabets match

$$E(\chi) = s_2 n_1 n_2$$

and

$$\sigma^2_\chi = n_1 n_2 \left[(n_1 + n_2) \left(s_3 - s_2^2 \right) + s_2^2 + s_2 - 2s_3 \right].$$

If they do not match then

$$E(\chi) = n_1 n_2 / q$$

and

$$\sigma^2_\chi = n_1 n_2 \left[(n_1 + n_2) \left(\frac{s_2}{q} - \frac{1}{q^3} \right) + \frac{1}{q} + \frac{1}{q^3} - \frac{2s_2}{q} \right].$$

[46] illustrates that the distributions of ϕ for properly and improperly matched alphabets overlap more than the distributions for χ properly and improperly matched. Thus it would be expected that the χ statistic would provide a better way of deciding whether two strings are from the same alphabet.

Incidence of Coincidence

The use of the ϕ statistic is closely connected the idea of coincidence. Given two random variables X_1 and X_2, then

$$P(X_1 = X_2) = \sum_{i=0}^{q-1} P(X_1 = i \cap X_2 = i)$$

is the probability of coincidence, κ. If the variables are independent then

$$\kappa = \sum_{i=0}^{q-1} (P(X = i))^2.$$

For random text

$$\kappa_r = \frac{1}{q},$$

and for a language with single letter frequencies as given in Appendix A, the probability of monographic coincidence in English plaintext is

$$\kappa_p = 0.066.$$

The distribution of the number of coincidences for text properly superimposed is binomial $\sim B(\kappa_p, k)$, where k is the total possible number of pairs.

A comprehensive practical review of five different test statistics for language recognition, using two letter and one letter probabilities, is given in [26].

4.2.2 Polyalphabetic Substitution Systems

A periodic polyalphabetic substitution system with period P is fully described by the first P substitutions in the sequence of substitutions. Since a polyalphabetic substitution system is simply a sequence of monoalphabetic substitutions, one method of cryptanalysing a polyalphabetic substitution system is to split the ciphertext into P separate strings and analyse each as a monoalphabetic substitution system. This requires knowledge of the period P.

Identification of the Period Using the ϕ Statistic

We wish to estimate parameter P using sample data y. One way to attempt this is use the MLE of P. That is to say we estimate P with $\hat{p}$, which is the value of $p = 1, 2 \ldots$ which maximises the probability of $\bar{\phi}$ taking a value at least as big as the one observed, assuming that each string has been subject to a monoalphabetic substitution. To calculate $\bar{\phi}$ we split the ciphertext into $p = 1, 2, \ldots$ subtexts, then

$$\bar{\phi} = \frac{\sum_{i=1}^{p} \phi_i}{p}.$$

So if A is the observed value of $\bar{\phi}$,

$$\begin{aligned} L(p|\bar{\phi} < A) &= P(\bar{\phi} < A|p) \\ &= P\left(Z < \frac{A - E(\phi)}{\frac{\sigma}{\sqrt{p}}}\right). \end{aligned}$$

Alternatively, we could employ the κ statistic, to look at the average number of coincidences between pairs of split ciphertext to achieve a similar result. [41] gives a technique for the automated cryptanalysis of polyalphabetic substitution systems with very small periods (up to 7), but generally, unless there is enough ciphertext to split the text into reasonably large monoalphabetic substitutions, cryptanalysis of substitution systems with ciphertext only is a difficult problem. However, the explicit specification of the key for a polyalphabetic system with a large period requires the listing of a huge amount of information. For example, the specification of a key for a polyalphabetic system with alphabet Z_{26} and period 100 requires the listing of 2600 mappings which then has to be transmitted securely to the intended receiver. To make this worthwhile the message would have to be at least 2600 letters

long, which would then provide the cryptanalyst enough text to attempt an attack. Because of this problem, polyalphabetic systems create a number of substitutions by the application of some function on a set of smaller substitutions. This provides a further area of possible weakness in a cryptosystem, because the substitutions thus generated are obviously interdependent, and if the cryptanalysist can find one he may be able to find the others. In the next section we describe just such a class of cryptographic systems.

4.3 Cryptanalysis of Rotor Machines

In the previous chapter we described a rotor machine as a periodic polyalphabetic substitution system, $E : \Sigma^+ \times K \rightarrow \Sigma^+$, defined on alphabet $\Sigma = Z_q$, where, $\forall k \in K$,

$$E(\cdot, k) =< S(R(i), k) \ \mid \ 0 \leq i < \infty > .$$

$R(i)$ is the set of known displacement functions and the key is the set of initial rotor substitutions, Π. The problem for the attacker with ciphertext only is to estimate Π given sample data y. A rotor machine with two rotors has a period 676, so the technique of splitting the ciphertext into monoalphabetically enciphered subtexts would require an inordinate amount of ciphertext to be effective. Rotor machines actually used had at least three rotors, with period at least 17576, and it was this large period that was thought to provide security.

Let

$$X = (X_0, X_1, \ldots, X_{m-1})$$

be a point in keyspace K. X_i is a permutation in S_q. We wish to find an estimate of Π, $\hat{X}$.

Iterative Technique

A permutation in S_q may also be described by a $q \times q$ *permutation matrix*. Let $S(R(i), X)$ be the permutation matrix for enciphering the i^{th} letter and $\mathbf{p}$ be the vector of single letter probabilities given in Appendix A. The probability of observing any particular string of ciphertext given a point in the keyspace is given by the likelihood function.

The Likelihood Function

The likelihood function is given by

$$L(X|y) = P(y_0|X)P(y_1|X)\cdots P(y_{n-1}|X).$$

The probability of observing a ciphertext letter is then given by

$$P(y_i|X) = e'_{y_i} S(R(i), X)\mathbf{p},$$

where e_{y_i} is the ${y_i}^{th}$ unit vector. So

$$L(X|y) = \prod_{i=0}^{n-1} e'_{y_i} S(R(i), X)\mathbf{p}.$$

The maximum likelihood estimate of Π is the point $X \in K$ for which $L(X|y)$ is maximum. The keyspace is too large to enumerate the likelihood function for all values of X. Instead we have to employ some search technique. The first method we look at involves imbedding the keyspace in the space of

stochastic matrices.

Noisy Rotors

A *column stochastic matrix* is a square matrix of non-negative elements with the property that the sum of the entries in each of the columns is one. We define Z_i as the set of all possible $q \times q$ column stochastic matrices, and

$$Y = (Y_0 \times Y_1 \times \cdots \times Y_{m-1})$$

as the space of all m-tuples of these matrices. Then $K \subset Z$. If we now let

$$X = (X_0, X_1, \ldots, X_{m-1})$$

be a point in Y, X can represent a bank of *noisy rotors*. A noisy rotor is a column stochastic matrix, where the elements of rotor X_i, x^i_{sr}, are no longer 0 or 1, but probabilities, where

$$x^i_{sr} = P(s \text{ is output for rotor i} | r \text{ is input for rotor i}).$$

We have embedded keyspace K in Y. We now wish to maximise the likelihood function

$$L(X|y) = \prod_{i=0}^{n-1} e'_{y_i} S(R(i), X)\mathbf{p}$$

over Y, with the anticipation that the estimate produced will be in the vicinity of Π. The MLE, $\hat{X}$, exists and is strongly consistent i.e.

$$\hat{X} \to \Pi \text{ as } n \to \infty$$

(see [1]). We now need some method to find $\hat{X}$. [1] suggests using an iterative algorithm based on the following result.

Theorem

Let F($\mathcal{Z}$) be a polynomial with non negative coefficients, homogeneous of degree d in its variables z_{ij}.

Let

$$\mathcal{Z} = \{z_{ij} : z_{ij} \geq 0, \sum_{i=0}^{q-1} z_{ij} = 1\},$$

and define the transformation $T : \mathcal{Z} \to \mathcal{Z}$ by

$$T(\mathcal{Z})_{ij} = \frac{z_{ij}\frac{\partial F}{\partial z_{ij}}}{\sum_{i=0}^{q-1} z_{ij}\frac{\partial F}{\partial z_{ij}}}$$

then $F(T(\mathcal{Z})) > F(\mathcal{Z})$ unless $T(\mathcal{Z}) = \mathcal{Z}$.

See [6] for a full theoretical treatment of the result. An m rotor system has a homogeneous likelihood function of degree mn. Since L is homogeneous and any $X \in Y$ satisfies the condition

$$X = \{x_{ij}^k : x_{ij}^k \geq 0, \sum_{i=1}^{m} x_{ij}^k = 1, 0 \leq k < m,\ 0 \leq j < q\}$$

by definition, the maximisation technique can be directly applied, with F replaced by L and $\mathcal{Z}$ replaced by X.

Calculating the Partial Derivatives

If we write

$$P(y_i|X) = f_i(x_{rs}^j)\ \ 0 \leq j < m,\ \ 0 \leq r, s < q$$

as simply f_i, then

$$L(X|y) = f_0 f_1 \cdots f_{n-1},$$

so

$$\begin{aligned}\frac{\partial L}{\partial x_{rs}^j} &= \frac{\partial f_0}{\partial x_{rs}^j} f_1 \cdots f_{n-1} + \frac{\partial f_1}{\partial x_{rs}^j} f_0 f_2 \cdots f_{n-1} \\ &\quad + \cdots + \frac{\partial f_{n-1}}{\partial x_{rs}^j} f_0 \cdots f_{n-2}.\end{aligned}$$

For any f_i, the variables x_{rs}^j are at most degree 1, so to calculate

$$\frac{\partial L}{\partial x_{rs}^j}$$

we need to calculate $f_0,\ f_1, \ldots,\ f_{n-1}$ and the coefficients of x_{rs}^j (this can be done at the same time by storing the coefficients of x_{rs}^j while calculating $f_0\ f_1 \cdots\ f_{n-1}$). Each f_i takes order mq^2 multiplications to calculate, so the partial derivatives take order nmq^2 multiplications.

An obvious starting point for the iterative algorithm is to set each X_{sr}^i to $1/q$. However, for systems with more than one rotor, there are many banks of rotors that are equivalent. Permutation

$$P_i C^{-a} C^a P_{i+1}$$

is the same as

$$P_i P_{i+1}.$$

That is to say there are multiple solutions to a rotor machine with more than one rotor. For example, with a three rotor machine, if we define the set of rotors

$$\rho = \{\rho_0, \rho_1, \rho_2\}$$

where

$$\rho_2 = \pi_2 C^{b_2},\ \ 0 \leq b_2 < q,$$

and

$$\rho_1 = C^{-b_2}\pi_1 C^{b_1}, \;\; 0 \leq b_1 < q$$

then if

$$\rho_0 = C^{-b_1}\pi_0,$$

$$\begin{aligned} S(R,\rho) &= C^{-r_2}\rho_2 C^{r_2} C^{-r_1}\rho_1 C^{r_1} C^{-r_0}\rho_0 C^{r_0} \\ &= C^{-r_2}\pi_2 C^{b_2} C^{r_2} C^{-r_1} C^{-b_2}\pi_1 C^{b_1} C^{r_1} C^{-r_0} C^{-b_1}\pi_0 C^{r_0} \\ &= C^{-r_2}\pi_2 C^{r_2} C^{-r_1}\pi_1 C^{r_1} C^{-r_0}\pi_0 C^{r_0} \\ &= S(R,\Pi). \end{aligned}$$

This means that we can arbitrarily fix a wiring in the last 2 rotors. This result can easily be extended to m rotors. So for the last $m-1$ rotors we are able to set one column to all zeros except for one entry, which is set to one. The rest of this row is set to zero also, and the remaining entries are initialised as $1/(q-1)$ to maintain the constraints.

Variations on the Technique

It seems reasonable to think that a technique which searched the space of *doubly stochastic* noisy rotors (i.e. the rows and columns sum to one) would perform better. However, there is no equivalent algorithm for doing so. One way of approximating a doubly stochastic algorithm is to alternate the iterations of the technique between column stochastic and row stochastic constraints. This tends to give better results, as Andelman showed in [1] and as our experimentation verified (see table 4.3).

n		2000	1500	1200	1000	800
plaintext 1	CS	75	121	*	*	*
	CS/RS	45	47	58	59	101
plaintext 2	CS/RS	57	53	61	70	*
plaintext 3	CS/RS	56	51	63	68	*
plaintext 4	CS/RS	63	71	73	85	*

Table 4.2: Number of iterations required with n letters of ciphertext for the iterative technique to solve the two rotor problem using the column stochastic technique (denoted CS) and the alternate column stochastic/row stochastic technique (denoted CS/RS).

Results of an Implementation of the Iterative Technique to Crack a 2 and 3 Rotor Machine

2-Rotor Machine

Table 4.3 gives the number of iterations required to solve a two rotor machine. A * indicates that the technique did not converge in 500 iterations.

3 Rotor Machine

We implemented the iterative technique for a three rotor system, using the rotors given in Appendix B, and a summary of the statistical properties of the 4 plaintext strings given in Appendix C. The algorithm found the correct three rotor machine for plaintext 1 with 4732 and 4056 letters in 394 and 589 iterations but with 3380 letters it had only achieved a correct decipherment rate (CDR) of 10% after 1000 iterations. For plaintext 2, 3 and 4, 1000 iterations (taking approximately 5 hours) gave a CDR of no better than 7%, little better than randomly guessing each letter, and half as good as guessing E for every letter. In chapter 5 we will see that a genetic algorithm performs

considerably better.

The Enigma Machine

The German Enigma was a three or four rotor machine fitted with a reflector. This had the effect of passing the text back through the rotors, thus making the enciphering and deciphering permutation identical. It was cracked because the Polish were able to recover the wirings of the machine through operational errors. This greatly simplified the problem for the English at Bletchley Park, but their success at decrypting many of the Germans' communications during World War 2 is nethertheless outstanding considering the limitation of the computational tools available to them. [35, 38] give detailed accounts of the story of how the Enigma was broken. In the next chapter we suggest some ideas as to how an Enigma machine with unknown rotors might be attacked.

4.4 Cryptanalysis and Block Ciphers

Block ciphers are substitution systems defined on Z_{2^t}, and t is chosen to make a statistical analysis of the 2^t letters infeasible. However, if we can find some dependence between some subset of the output positions and input positions then a statistical attack may be possible. Given a block cryptosystem, B, and ciphertext, y,

$$y = B(x, k),$$

if we write $x = (x_0, x_1, \ldots, x_{2^t-1})$ and $y = (y_0, y_1, \ldots, y_{2^t-1})$, where x_i is the i^{th} input bit and y_i is the i^{th} output bit, and let y'_s be a subsequence of s

output bits and x'_t be a subsequence of t input bits then we are interested in seeing if there is some dependence between the observed y'_s values and the x'_t values. For example, suppose $s = 3, t = 3$, and

$$y'_s = (y_0, y_3, y_{17}) \text{ and } x'_t = (x_0, x_1, x_2),$$

then we may be interested in finding out whether, for some fixed key, k, certain y'_s values are more likely to occur when x has certain x'_t values. Alternatively we may wish to test, for some fixed x, whether certain y'_s values are more likely to occur when k has certain k'_t values, where k'_t is a subsequence of t key bits.

This type of dependence can be checked using the χ^2 test of fit. The χ^2 distribution is a special case of the gamma distribution with $\alpha = v/2$ and $\beta = 2$, where parameter v is a positive integer, called the degrees of freedom. A random variable X has a χ^2_v *distribution* iff its probability distribution is given by

$$f(x) = \begin{cases} \frac{1}{2^{v/2}\Gamma(v/2)} x^{v/2-1} e^{-\frac{x}{2}} & \text{for } x > 0 \\ 0 & \text{elsewhere,} \end{cases},$$

Let $(X_1, X_2, \ldots, X_l)$ be identically distributed random variables with range Z_q and unknown probability distribution. Let p be a probability distribution defined on Z_q. The χ^2 test is used to test the hypothesis that

$H_0 : p$ is the probability distribution of $(X_1, X_2, \ldots, X_l)$

against

$H_1 : p$ is not the probability distribution of $(X_1, X_2, \ldots, X_l)$

The X^2-statistic is the random variable defined as

$$X^2 = \sum_{i=0}^{q-1} \frac{(n_i - n\ p(i))^2}{n\ p(i)}.$$

Pearson [60] showed that, if p is the common distribution of $(X_1, X_2, \ldots, X_l)$, the limiting form of the distribution of X^2 is the χ^2 distribution with $l-1$ degrees of freedom, so a one or two sided hypothesis test using the tail ends of the χ^2 distribution can be constructed. Other tests of fit include the Likelihood Ratio test and the Kolmogorov-Smirnoff test. See [77] for a full treatment of tests of fit and [45] for a description of their use in cryptography.

For a block cipher, we are interested in testing whether either of the following holds:

- some set of output bit coordinates, y'_s, are dependent on some set of input bit coordinates, x'_t, for a fixed key, k, or
- some set of output bit coordinates, y'_s, are dependent on some set of key bit coordinates,k'_t, for a fixed input x.

If we define random variables Y'_s over Z_{2^s} (the possible values of y'_s), X'_t over Z_{2^t} (the possible values of k'_s) and K'_t over Z_{2^t} (the possible values of k'_s), then the above dependencies can be tested with the following hypotheses

- We wish to test

$$\begin{aligned} H_0 \quad &: \quad P(Y'_s = i \cap X'_t = j|k) = 2^{-s}2^{-t},\ 0 \le i < 2^s,\ 0 \le j < 2^t \\ &\quad\ against \\ H_1 \quad &: \quad P(Y'_s = i \cap X'_t = j|k) \neq 2^{-s}2^{-t},\ 0 \le i < 2^s,\ 0 \le j < 2^t. \end{aligned}$$

- We wish to test

$$H_0 \quad : \quad P(Y'_s = i \cap K'_t = j|x) = 2^{-s}2^{-t},\ 0 \leq i < 2^s,\ 0 \leq j < 2^t$$

$$against$$

$$H_1 \quad : \quad P(Y'_s = i \cap K'_t = j|x) \neq 2^{-s}2^{-t},\ 0 \leq i < 2^s,\ 0 \leq j < 2^t$$

For the first test, take a random sample of plaintext letters, of size n, and a random key, k, and encipher. Calculate the observed instances of each of the possible values of Y'_s and X'_t . Let $N(i,j)$ denote the number of times $Y'_s = i$ and $X'_t = j$. Calculate the X^2 statistic,

$$X^2 = \sum_{i=0}^{2^s-1} \sum_{j=0}^{2^t-1} \frac{(N(i,j) - n2^{-(s+t)})^2}{n2^{-(s+t)}},$$

and compare the value with the $\chi^2_{2^{-(s+t)}-1}$ value, for the significance level of the test, from tables. H_0 is rejected if $X^2 > \chi^2_{2^{-(s+t)}-1}$. A similar procedure is followed to test the second hypothesis. Obviously there are too many possible dependencies to test them all, but tests of fit on DES were performed extensively by IBM and the NSA when the cryptosystem was introduced, and no obvious dependencies were found.

Differential Cryptanalysis

Although there may be no obvious dependencies between plaintext and ciphertext subsequences, it has been shown that there is dependency between *differences* in plaintext letters and differences in ciphertext letters. First made public by Eli Biham and Adi Shamir in 1990, differential cryptanalysis is a method which searches for the maximum likelihood estimator

of a key, where the likelihood function is derived from the effect of particular differences in plaintext pairs on the differences of the associated ciphertext pairs. It was developed to attack DES like cryptosystems. The technique is described in [8]. The difference between two text blocks, X_1, X_2, is the result of XORing X_1 and X_2,

$$\Delta X = X_1 + X_2.$$

In Bayesian terms, differential cryptanalysis can be outlined as follows: Initially, you have no prior knowledge as to the correct key. Your prior distribution is uniform over the keyspace. Choose a pair of plaintext with some fixed difference and encrypt. The difference in the resulting ciphertext modifies your belief as to the correct key, since certain keys are more likely to produce certain differences than others. Continue to encipher plaintext pairs with the fixed difference until your prior distribution exhibits a clear maximum. The best attack against 16-round DES requires 2^{47} chosen plaintexts (or 2^{55} known plaintexts). Differential cryptanalysis is essentially a theoretical attack, because of the huge amounts of processing time and data requirements. If you can choose 2^{47} different plaintexts then you can read a large amount of your opponent's messages!

Chapter 5

Genetic Algorithms for Permutation Problems

5.1 Introduction

Genetic algorithms (GAs) were first proposed by John Holland [36], whose ideas were applied and expanded on by Goldberg [27]. GAs are a heuristic search technique based on the principles of the Darwinian idea of survival of the fittest and natural genetics. Holland's work was primarily an attempt to mathematically understand the adaptive processes of nature, but the general emphasis of GA research since then has been in finding applications, many in the field of combinatorial optimisation. In this field GAs must be seen as one of many possible heuristic techniques, many of which are described in [66].

5.2 The Algorithm

GAs act on a population pool of proposed solutions, called *chromosomes*. The pool is generally maintained to be of constant size, p. Each proposed solution is evaluated using a fitness function. The initial population pool is generally randomly selected. At each *generation* a possibly variable number of chromosome pairs are *selected* to act as *parents*. From each pair two *offspring* are *created* using specified genetic operators. The new collection of offspring chromosomes are then *merged* back into the population pool in a way that maintains the constant size, p. This process continues until some specified finishing criterion is satisfied. For a more detailed description of GAs see [27]. The decisions in implementing a GA for a particular problem are concerned with representation, fitness evaluation and the select, create and merge methods. The algorithm as described in [74] is given in figure 5.1.

An o-schema is a set of permutations with a particular set of elements in common positions. For example, the permutation 1 4 2 3 5 is an instance of o-schema 1 * * * 5, where * can be any element not already specified. Two o-schemata are said to be *compatible* if it is possible for a chromosome to be an instance of both o-schemata. For example, 1 * * * 5 and 1 3 * * * are compatible, but 1 * * * 5 and 2 3 * * * are not. GAs act to combine short o-schemata with high average fitness to produce better solutions. Hence the type of problem on which a GA will be effective is one where "building blocks" can be combined to construct good solutions. Holland discusses in depth the reasons why GAs work.

```
GA
    {objective is to maximise value(p) such
    that p ∈ U, the universal solution set }
    P, Q, R : multi-set of solutions ⊂ U;
    initialise(P);
    while not finish(P) do
        begin
        Q := select(P);
        R := create(Q);
        P := merge(P, Q, R)
        end
end GA
```

Figure 5.1: The genetic algorithm paradigm

5.3 Representation

The most obvious representation of a permutation is an array of q integers. The elements of a permutation will be indexed from 1 to q, so that $P[i]$ will denote the i^{th} element of a permutation, P. The problem with this representation is that the traditional crossover techniques will not produce permutations. In the section on create, various operators for the permutation representation are discussed, but other representations have also been proposed. Among these are the ordinal and adjacency representation [32], the positions listing representation [62] and the precedence matrix representation [25]. The ordinal and adjacency represention are generally accepted

to be unsatisfactory for reasons given in [32] and are not covered here. The position listing representation for any permutation, P, is simply the inverse, P^{-1}. The precedence matrix representation for a permutation is a $q \times q$ matrix, the manipulation of which is obviously more time consuming than operations involving a permutation. For each element of a permutation, P, the precedence matrix, M, has an entry 1 in position $M[x, y]$ if and only if element x occurs before element y in permutation P. Otherwise the entry is 0.

5.4 Select Methods

The purpose of a select procedure is first to generate a discrete probability distribution function (pdf) over the population pool which is then randomly sampled for each parent required. The number of parents required at each generation is generally either fixed or an even random number between 2 and p. It is necessary that the better (higher if maximising and lower if minimising) the fitness of a chromosome the higher probability it should have of being chosen. Two methods of generating a pdf are described below.

Roulette

Proposed by Goldberg in [27], roulette is the most commonly used selection procedure. Given a population of size p, the relative fitness of each chromosome is evaluated and used as the probability of selection. Let X be a random variable defined on the population pool, and $f_i \in \Re^+$ denote the

fitness of the i^{th} member of the pool, x_i. Then, for maximisation problems,

$$P(X = x_i) = \frac{f_i}{\sum_{i=1}^{p} f_i},$$

is the probability of selecting chromosome x_i. For minimisation problems the relative deviation from the maximum current fitness value is used as the fitness. Let f_{max} be the largest fitness value in the current population. Let

$$f_i' = f_{max} - f_i,$$

then the probability of selecting chromosome x_i is

$$P(X = x_i) = \frac{f_i'}{\sum_{i=1}^{p} f_i'}.$$

Variations of the basic roulette procedure have been suggested [4].

Ranking

This method was introduced by Baker in [3]. Selection by ranking shifts the emphasis onto a prospective parents' relative position in the population pool irrespective of actual fitness values. Let p_i denote the rank of chromosome x_i in the population pool, then the probability distribution used by Baker is defined as

$$P(X = x_i) = \frac{2p_i}{p(p+1)}.$$

An alternative distribution is given by

$$R_i = \frac{(p - p_i)^r}{\sum_{j=1}^{p} j^r},$$

$$P(X = x_i) = \frac{R_i}{\sum_{j=1}^{p} R_j},$$

where r is a ranking parameter, which allows the weighting of importance of position. So $r = 0$ is equivalent to random selection, and setting $r > 1$ increases the probability of the top chromosomes being selected.

Another popular selection method, particularly in parallel GA implementations, is tournament selection, described in [57]. Other selection methods include Fibonacci, exponential and tendency, all of which are simply different ways of ranking the population pool, and exclusive, which only selects chromosomes from the top of the population pool.

5.5 Create Method

The create stage of the algorithm requires defining a method of generating offspring of any two parents selected for mating which are also valid solutions. This is achieved using a *crossover operator* and *mutation*. Apart from the ordinal representation, all of the representations mentioned above require specialised crossover operators. The *alternating edges* and *subtour chunking* operators have been proposed for the adjacency representation [32], two *tie breaking* operators for the positions listing representation [62] and the *intersection* and *union* crossover operators for the precedence matrix representation [25]. In this section only operators for the permutation representation are described in detail. The operators described below are also valid for chromosomes defined by the positions listing representation, since these chromosomes are also permutations. Conversely, the tie breaking operators described by Poon and Carter [62] are valid operators for the permutation

representation. It is a problem specific issue as to which representation and operator to use. The tie breaking operators create a child by two point crossover and replace any duplicates with the missing elements randomly. Intersection crossover preserves the precedence relationships which are common to both parents and union crossover combines some precedence information from both parents. Poon and Carter give a method of performing union crossover without having to use the precedence matrix representation.

5.5.1 Crossover Operators

A good operator will combine the information from both parents with as little distortion as possible. If the permutation representation is used, the traditional methods of crossover used for combinatorial problems such as one point, two point and uniform cannot be applied, since the offspring will generally not be permutations. For this reason several alternative operators have been suggested. Radcliffe [65] describes the desirable properties of genetic operators in the general case. These properties, in relation to permutation problems, are reproduced below, but in assessing the usefulness of any permutation operator it is important to consider the nature of the problem. In some permutation problems the absolute position of the elements is the overriding concern. For example, in job-shop scheduling [23], the elements represent jobs and their position on the chromosome represents the machine on which to perform the job and the order in which the job should be done. In other permutation problems, position relative to the other elements may be of more importance. In [79] it is argued that this is true of TSP. Radcliffes' informal analogy is reproduced here to help explain the properties.

Suppose the chromosomes are people and characteristics used to define a set of o-schemata are hair colour and eye colour.

- Respect. An operator is respectful to both parents iff for parents P_1, P_2 and offspring C_1, C_2,

$$P_1[i] = P_2[i] \Rightarrow C_1[i] = C_2[i] = P_1[i] \quad 0 < i \leq q.$$

  If both parents have blue eyes then their offspring must have blue eyes.

- Strictly Transmit. An operator strictly transmits iff, $0 < i \leq q$,

$$C_1[i] = P_1[i] \text{ or } C_1[i] = P_2[i], \text{ and } C_2[i] = P_1[i] \text{ or } C_2[i] = P_2[i], .$$

  If one parent has blue eyes and brown hair and the other has brown eyes and red hair, the child must have either blue or brown eyes and either brown or red hair.

- Ergodicity. An operator is ergodic iff, for any set of chromosomes, it is possible to access any point in the search space through the finite application of the operator. If the whole population has blue eyes, it must still be possible to produce a brown eyed child. This is a property that is desirable in a mutation operator.

- Properly Assort. An operator properly assorts iff, given two instances of compatible o-schemata, it is possible for the operator to produce an offspring which is an instance of both o-schema. i.e. for any subsets of elements of the two parents with no elements in contradictory positions, it is possible for the offspring to have both these subsequences.

So, for example, if $P_1 = (1, 2, 4, 3)$ and $P_2 = (3, 2, 4, 1)$, it must be possible to have offspring $C_1 \in$ o-schema $(1, *, 4, *)$ and $C_1 \in$ o-schema $(*, 2, *, *)$, or any other compatible o-schemats for which the parents are instances. If one parent has blue eyes and the other brown hair it must be possible to recombine them to produce a child with blue eyes and brown hair as a result of the crossover. The only permutation operator we have found which properly assorts is Radcliffes' R^3 operator (random respectful recombination), which, as the name suggests, copies the common elements of both parents to the child and fills the remaining positions randomly. The R^3 operator is presented more as a useful starting point from which to design new operators than as a useful tool in itself.

The crossover operators for permutation problems described here are given in [76].

Partially Mapped Crossover (PMX)

PMX was first proposed by Goldberg and Lingle [28]. PMX maps a randomly selected section of one parent directly onto an offspring by a sequence of transpositions on the other parent. The PMX algorithm is given in figure 5.2. PMX can be written in terms of transpositions on a parent. A transposition is a permutation where two elements are swapped and the rest remain in the same position. Using cycle notation, a transposition swapping a and b is denoted $(a\ b)$. The transposition (a, a) is the identity. Let k be the size of

Input arrays P_1 and P_2;
    $P_1 := C_1$;
    $P_2 := C_2$;
    generate two random numbers $0 < m_1 < m_2 \leq q$.;
    for $i = m_1$ to m_2 **do**
        temp := position of $P_2[i]$ in P_1;
        swap elements in positions i and temp in C_1;
        temp := position of $P_1[i]$ in P_2;
        swap elements in positions i and temp in C_2;
output arrays C_1 and C_2;

Figure 5.2: Partially mapped crossover

the mapping section, $k = m_2 - m_1 + 1$. Let

$$\begin{array}{rcccc} t_1 & = & (& m_1 & P_1^{-1}(P_2(m_1)) \quad) \\ t_2 & = & (& m_1+1 & (P_1 \circ t_1)^{-1}(P_2(m_1+1)) \quad) \\ \vdots & & & \vdots & \vdots \\ t_k & = & (& m_2 & (P_1 \circ t_1 \circ \cdots \circ t_{k-1})^{-1}(P_2(m_2)) \quad), \end{array}$$

then

$$C_1 = P_1 \circ t_1 \circ \cdots \circ t_k.$$

C_2 is found by reversing P_1 and P_2. To summarise PMX,

1. copy parent 1 and parent 2 to child 1 and child 2 respectively.

2. Generate a random *mapping section*, i.e., two random numbers $0 < m_1 < m_2 \leq q$.

3. In child 1 swap the element in map 1 with the element in position map 1 of child 2, and vice versa.

4. Repeat for all positions up to map 2.

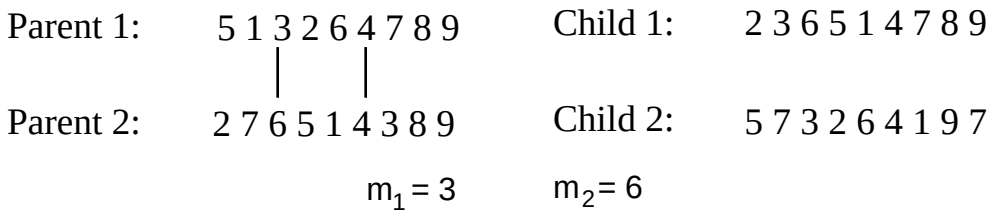

Figure 5.3: An example of PMX

So, for the example in figure 5.3, PMX proceeds as follows:

- Copy parent 1 and parent 2 to child 1 and child 2 respectively.
- The mapping section is positions 3, 4, 5 and 6.
- Swap elements 3 and 6 in child 1. Child 1 is now 5 1 6 2 3 4 7 8 9.
- Swap elements 2 and 5 in child 1. Child 1 is now 2 1 6 5 3 4 7 8 9.
- Swap elements 1 and 3 in child 1. Child 1 is now 2 3 6 5 1 4 7 8 9.
- Element 4 remains in the same position.
- Repeat for child 2.

PMX is respectful, but does not strictly transmit. This is illustrated in the above example, where $C_1[2] = 2$ but $P_1[2] = 1$ and $P_2[2] = 7$.

Cycle Crossover

Proposed in [59], cycle crossover is respectful and strictly transmits, i.e. it preserves the position of the elements from one parent or the other. A starting point, s_0, is selected at random. Child 1 inherits the element in the starting position of Parent 1, i.e. $C_1[s_0] := P_1[s_0]$. The element in $P_2[s_0]$ is copied into the position, s_1, for which $P_1[s_1] = P_2[s_0]$. The element in $P_2[s_1]$ is copied into the position, s_2, for which $P_1[s_2] = P_2[s_1]$. This continues until $P_2[s_k] = P_1[s_0]$, and the cycle is completed. The remaining elements of C_1 are inherited from P_2. Cycle crossover is presented in 5.4.

Cycle crossover can also be described by a permutation on each parent. Given $s_0, 0 < s_0 < q$, let

$$\begin{aligned} s_1 &= P_2^{-1}(P_1(s_0)) \\ s_2 &= P_2^{-1}(P_1(s_1)) \\ \vdots & \qquad \vdots \\ s_k &= P_2^{-1}(P_1(s_{k-1})), \end{aligned}$$

where $s_{k+1} = P_2^{-1}(P_1(s_k)) = s_0$, then cycle crossover is achieved by composing P_2 and the cycle permutation $(s_0\ s_1\ \cdots\ s_k)$, i.e.

$$C_1 = P_2 \circ (s_0\ s_1\ \cdots\ s_k).$$

C_2 is found by reversing P_1 and P_2 to get a different cycle permutation. For the example in figure 5.5, child 1 inherits elements 1, 3, 6 and 7 from parent 1 and 2, 5, 4, 8 and 9 from parent 2.

Input arrays P_1 and P_2;

$C_1 := P_2$;

$C_2 := P_1$;

randomly select $s_0,\ 0 < s_0 \leq q$;

$s_k := s_0$;

$C_1[s_0] := P_1[s_0]$;

while $P_2[s_k] \neq P_1[s_0]$ **do**

$temp :=$ position of $P_2[s_k]$ in P_1;

$C_1[temp] := P_2[s_k]$;

$s_k := temp$;

$s_k := s_0$;

$C_2[s_0] := P_2[s_0]$;

while $P_1[s_k] \neq P_2[s_0]$ **do**

$temp :=$position of $P_1[s_k]$ in P_2;

$C_2[temp] := P_1[s_k]$;

$s_k := temp$;

Output C_1 and C_2;

Figure 5.4: Cycle crossover

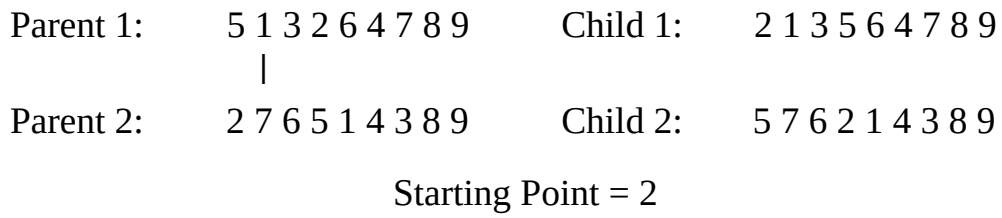

Figure 5.5: An example of cycle crossover

Order Crossover, Version 1

Order crossover was proposed in [15]. As with PMX a mapping section is randomly selected and the elements within this section of Parent 1 are copied to Child 1. The remaining positions in Child 1 are found from Parent 2 in the order they appear in that parent, ignoring any elements already included in Child 1. The process is reversed to find the second offspring. Figure 5.7 illustrates that order crossover is not respectful nor does it strictly transmit, as $P_1[8] = P_2[8] = 8$ but $C_1[8] = 9$ (an operator cannot strictly transmit and not be respectful).

The order 1 permutation can be derived as follows:

given, m_1, m_2, where $0 < m_1 < m_2 \leq q$, let $P_1' = P_1 \circ C^{m_2}, P_2' = P_2 \circ C^{m_2}$, where C is the shift permutation, $(1\ 2\ \cdots\ q)$ in cycle notation, and

$$\begin{aligned} C^1 &= C \\ C^k &= C^{k-1}. \end{aligned}$$

Input arrays P_1 and P_2;

randomly select m_1 and m_2 such that $0 < m_1 < m_2 \leq q$;

for $i = m_1$ to m_2 **do**

$C_1[i] := P_1[i]$;

$EmptyPositions = q - 1 - m_2 + m_1$;

let $M = \{P_1[m_1], P_1[m_1 + 1], \ldots, P_1[m_2]\}$;

$i := 0$;

$j := m_2$;

while $(i < EmptyPositions)$ **do**

if $P_2[j \bmod q + 1] \notin M$ **do**

$C_1[(i + m_2) \bmod q + 1] := P_2[j \bmod q + 1]$;

$i := i + 1$;

$j := j + 1$;

Output C_1;

Figure 5.6: Order crossover, version 1, for one offspring

Let $P = \{p_1, p_2, \ldots, p_{m_1 - 1}\}$, where

$$\begin{aligned} p_1 &= (P_2')^{-1}(P_1'(1)) \\ p_2 &= (P_2')^{-1}(P_1'(2)) \\ &\vdots \qquad\qquad \vdots \\ p_{m_1-1} &= (P_2')^{-1}(P_1'(m_1 - 1)). \end{aligned}$$

Let $N =< n_1, n_2, \ldots, n_{m_1} >$, where N is the ordered sequence of the elements

of P. If O_1 is a permutation in S_q, where

$$\begin{aligned} O_1(i) &= (P_2')^{-1}(P_1'(n_i)) \quad \text{if } i < m_1 \\ O_1(i) &= i \qquad\qquad\qquad\quad \text{otherwise} \end{aligned}$$

then

$$C_2 = P_2 \circ O_1 \circ C^{q-m_2}.$$

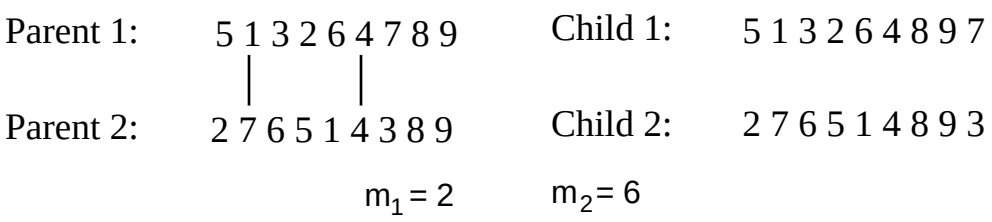

Figure 5.7: An example of order crossover, version 1

Order Crossover Version 2

Originating in [78], this alternative order based crossover takes a fixed number of random positions, and imposes the order of the elements in these positions in one parent on the other parent to produce a child. Order 2 crossover can also be presented as a permutation. Given $N =< n_1, n_2, \ldots, n_k >$, where $0 < k < q$ and $0 < n_1 < n_2 < \ldots < n_k \leq q$, let $R = \{r_1, r_2, \ldots, r_k\}$, where

$$\begin{aligned} r_1 &= P_2^{-1}(P_1(n_1)) \\ r_2 &= P_2^{-1}(P_1(n_2)) \\ \vdots &\qquad\qquad \vdots \\ r_k &= P_2^{-1}(P_1(n_k)). \end{aligned}$$

Let $R' =< r'_1, r'_2, \ldots, r'_k >$, where $r'_i \in R, r_i \neq r_j, 0 < i, j \leq k, i \neq j$ and $r'_1 < r'_2 < \ldots < r'_k$. The order permutation, O_2, is defined as

$$\begin{aligned} O_2(i) &= r'_i \quad \text{if } i \in R, \\ O_2(i) &= i \quad \text{othewise.} \end{aligned}$$

So

$$C_2 = P_2 \circ O_2.$$

Order 2 crossover does not strictly transmit.

Input arrays P_1 and P_2;

$C_1 := P_1$;

randomly select k positions, $< n_1, n_2, \ldots, n_k >$,

where $n_1 < n_2 < \cdots < n_k$;

Find the sequence $R =< r_1, r_2, \ldots, r_k >$,

where $P_1[r_i] = P_2[n_i]$;

Sort the sequence R;

let $C_1[r_1] = P_2[n_1], C_1[r_2] = P_2[n_2], \ldots, C_1[r_k] = P_2[n_k]$;

Output C_1;

Figure 5.8: Order crossover, version 2, for one offspring

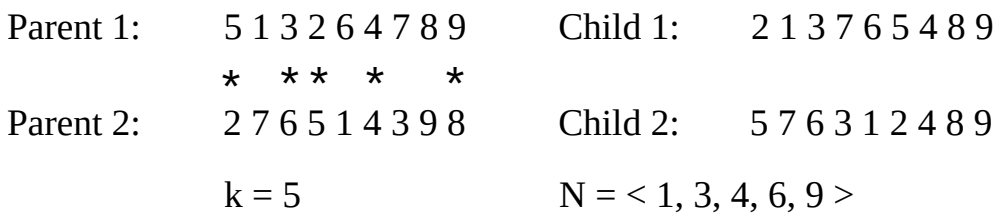

Figure 5.9: An example of order crossover, version 2

Position Based Crossover

Also proposed by Syswerda [78], position based crossover is very similar to the first version of order crossover except for the fact that the mapping section is a fixed number of random positions rather than a randomly selected section. Given $R =< r_1, r_2, \ldots, r_k >$, where $0 < r_1 < r_2 < \cdots < r_k \leq q$, let $P = \{p_1, p_2, \ldots, p_{m_1 - 1}\}$, where

$$\begin{aligned} p_1 &= (P_2)^{-1}(P_1(r_1)) \\ p_2 &= (P_2)^{-1}(P_1(r_2)) \\ &\vdots \qquad\qquad \vdots \\ p_{r_k} &= (P_2)^{-1}(P_1(r_k)). \end{aligned}$$

Let $N =< n_1, n_2, \ldots, n_{m_1} >$, where N is the ordered sequence of the elements of P. If B is a permutation in S_q, where

$$\begin{aligned} B(i) &= (P_2)^{-1}(P_1(n_i)) \quad \text{if } i \in R \\ B(i) &= i \qquad\qquad\qquad\quad \text{otherwise} \end{aligned}$$

then

$$C_2 = P_2 \circ B.$$

Edge Recombination

Edge recombination, proposed in [78], and the enhanced version, proposed in [76], were designed for problems like the TSP where relative position of the nodes is of more importance than absolute postition. Both produce only one offspring. The adjacency information is summarised in an edge list, which, for every element, lists the adjacent elements in both parents. An offspring is produced as follows:

- Select a starting element, s_0. This is generally the first element in either parent.
- Remove all entries in the edge table for s_0. The next element in the offspring is chosen from the elements that have links to s_0, i.e. the elements that have entries in the row for s_0. The next element, s_1, is the one with fewest entries remaining in its row of the edge table. If there is a tie for the least number of entries, s_1 is selected randomly from the elements tieing.
- Repeat the previous step until the offspring is complete.

So for the example in figure 5.10, 5 was chosen as the starting element. After 5 has been removed from all rows, of the three elements connected to 5, 1 and 9 have two entries and 6 has three. In this case 9 was randomly chosen. The row for 9 has elements 8 and 2 left, which have three and two elements in their respective rows, so 8 is chosen and removed form the list. The choice now is between 7 and 3 both of which have three elements. 3 was chosen. 1 is the next element as by now it only has one element left in the

Parent 1: 5 1 3 2 6 4 7 8 9 Offspring 5 9 8 3 1 4 6 7 2

Parent 2: 2 7 6 5 1 4 3 8 9 Starting Node = 5

Edge Table:

Element	Links
1	5, 3, 4
2	3, 6, 9, 7
3	1, 2, 4, 8
4	6, 7, 1, 3
5	9, 1, 6
6	2, 4, 7, 5
7	4, 8, 2, 6
8	7, 9, 3
9	8, 5, 2

Figure 5.10: An example of edge recombination

list, which means 4 must be next. 4 offers either 6 or 7. 6 gives 2 or 7 and finally 2 must be the last element.

As defined above the operator is not concerned with the direction of the edge. In the above example the common edge between 8 and 9 is maintained but the elements are reversed.

Enhanced Edge Recombination

While the original edge recombination maintains common edges, the fact that it ignores edge direction means it may well disrupt common subsequences. The enhanced edge recombination operator was designed to preserve these common subsequences. When constrcucing the edge list, if both parents share a common edge this edge is flagged in the edge list by making the entry nega-

tive. Entries with negative values are then given priority in the construction of offspring.

Union Crossover

A random substring from parent 1 is selected and stored in a list, S_1, and the remaining unused elements are listed separately in the order they appear in parent 2 in list S_2. An offspring is constructed as follows: randomly select either S_1 or S_2. Write the first element of the chosen substring to offspring child 1 and delete it from the chosen substring. If the substring is not empty, repeat to find the next element of child 1. When one of the substrings is empty, copy the elementds remaining in the non empty substring to the child. Child 2 is found by reversing the role of parent1 and parent 2.

Operator	Respectful	Strictly Transmits	Properly Assorts
PMX	Yes	No	No
Cycle	Yes	Yes	No
Order 1	No	No	No
Order 2	No	No	No
Position	No	No	No
Edge Rec	No	No	No
Enhanced	No	No	No
Union	No	No	No

Table 5.1: A summary of the properties of the genetic operators

5.5.2 Mutation

Mutation is important to maintain diversity within the population pool. The two commonest forms of mutation of permutations are

1. **Swap.** Randomly select two positions and swap the elements in those positions.

2. **Shift.** Randomly select an element and put that element in a random position, shifting the other elements right. An alternative form is to select a random subsequence and insert this subsequence in a random position (see figure 5.11).

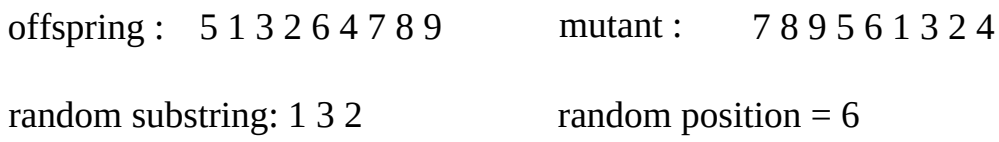

Figure 5.11: An example of shift mutation

Both mutation operators are ergodic. The usefulness of judging the operators by the classifications used above is strictly limited. For example, of those described above cycle crossover is the only operator that strictly transmits, but was found in [76] to be the least effective for solving the TSP. When choosing an operator for a permutation problem it is vital to consider what information is central in building a good solution. In some problems both position and edges may play a part, as is the case in the rotor problem considered in the next chapter. In such cases some form of hybridised edge/position operator could be used. As will be discussed in chapter 6, it

was found that an increased use of the shift mutation operator can also be effective.

5.6 Merge Method

Introducing the offspring to the population pool requires the removal of some or all of the existing chromosomes to maintain a constant pool size. Holland's original idea was to replace all of the previous generation with the offspring (this obviously requires creating p offspring). He also suggested that offspring could randomly replace members of the population pool. This has the advantage of maintaining diversity, but means a loss of information as fit chromosomes are often removed. The *Best fit* merge method only allows an offspring into the pool if it has fitness better than the current worst solution in the pool, which it then replaces. De Jong introduced the idea of *elitist* merge mechanisms, where a certain number of the best solutions in the population pool are always maintained [18]. The method termed *new solutions* is to only allow offspring into the pool if they had better fitness than the worst member of the current pool and if they were not simply replicas of existing pool members. An offspring to be included replaces the chromosome whose fitness is closest to and less than (when maximizing) the fitness of the offspring. The new solutions method helps to prevent premature convergence by stopping the duplication of solutions already in the population. *Replace* merge mechanism accepts offspring into the population if their fitness is better than the mean fitness (or some multiple of the mean fitness), which replace the worst current members of the population. The *tendency*

merge method removes as many chromosomes from the population as there offspring in the reproduction pool, and selects the offspring to merge in a way that favours the best of the reproduction pool.

Chapter 6

Cryptanalysis of Rotor Machines using a Genetic Algorithm

6.1 Introduction

GAs have been used on a wide range of problems involving the discrete search space of combinatorial optimisation problems. The type of problem on which they are successful has been described as problems where the idea of building blocks is relevant (see chapter 5). If a GA is being used to search a keyspace for a cryptosystem, then this concept of building blocks is equivalent to a dependency between subblocks of key and ciphertext. GAs have previously been used in cryptanalysis for solving simple substitution systems [22], transposition ciphers [56] and knapsack based systems [75].

The crytographic systems solved by a GA have, upto now, all been fairly straightforward, and readily solvable by other methods. The knapsack system solved by Spillman [22] involved solving a fifteen item knapsack problem. We are interested in exploring whether GAs are useful in cracking more difficult cryptosystems. More specifically we concentrate on their use in cracking rotor machines, which are polyalphabetic substitution systems with reasonably large periods.

We show that a genetic algorithm can be used to successfully search the very large discrete keyspace of a rotor machine, of magnitude up to $(26!)^3$, using a simple statistical measure of fitness, and find that this method is superior to the method described in the previous chapter. The first method used searches the whole rotor space and uses the likelihood as a fitness measure. The second method involves finding the last $m-2$ rotors of an m rotor machine using a GA, with a fitness measure based on the phi statistic, then solving the resulting two rotor machine using the iterative technique described in chapter 4. The GA implementation is described in section 2, and both fitness measures are described in detail in section 3. The plaintext is assumed to be n independent realisations of a random variable defined on alphabet Z_q, with probability distribution derived from observation of the single letter frequencies of English. The distribution we used is that given in [45] and is given in Appendix A. The success or otherwise of the cryptanalysis of two, three and four rotor machines using the two GAs is detailed in section 4.

6.2 The GAmeter Toolkit

The GAmeter toolkit and the X-windows version, X-GAmeter, is a platform for implementing genetic algorithms on a wide range of problems developed at the UEA by A. Kapsalis and J. Mann [54, 40, 39]. The version used for this problem (version 1.5) was not constructed for permutation problems, and so a large number of alterations were required. The stopping condition used in all experiments is that the GA will stop when there has been no improvement in the best solution found to date in 500 generations. The next release of GAmeter will have integrated permutation and sequencing problems fully into the framework.

6.2.1 Representation

The initial motivation of this thesis was to attempt to use a genetic algorithm to search the space of noisy rotors (see chapter 4) using the likelihood function as a fitness measure. However, it was felt that it would be worth exploring using a genetic algorithm to search the discrete key space also. The noisy rotor representation has the advantage of allowing traditional crossover techniques, but requires very large chromosomes and the fitness evaluation is very slow. After initial experimentation with the noisy representation and the permutation representation, we decided to continue with the latter only. The positions listing, or inverse, representation was used for comparison purposes on a three and four rotor machine, and while the precedence matrix representation is not explicitly used, the permutation version of union crossover was experimented with. Throughout this chapter a correct set of rotor solu-

tions will be denoted ρ, where $\rho = \{\rho_0, \rho_1, \ldots, \rho_{m-1}\}$, and a proposed set of solutions ρ', where $\rho' = \{\rho'_0, \rho'_1, \ldots, \rho'_{m-1}\}$.

GAmeter stores a bitstring chromosome as an array of 32 bit integers. The most obvious way of storing a permutation chromosome is as an array of integers representing each element, and, in order to fit a permutation problem into the GAmeter framework, it is necessary to continually translate parts of the chromosome into integers. Although GAmeter's representation saves memory this is probably offset by the work done in the translation. However, changing the representation would effectively require the rewriting of GAmeter, so we opted to continue with the bitstring chromosomes.

6.2.2 Select Methods

The selection methods available on GAmeter are: random, roulette, Fibonacci, Exponential, tendency and exclusive. Except in the early stages of the research, when ranking was employed, roulette selection was used exclusively. Roulette was found to produce faster convergence to an optimal solution, although the difference was minimal.

6.2.3 Create Methods

PMX, cycle crossover, both versions of order crossover, position based crossover, union crossover and enhanced edge recombination were all added to GAmeter. Shift and both forms of swap mutation also had to be coded. Most of the experiments were performed using PMX, since it worked so well. The difficulty of the problem was judged by two parameters: the number of rotors and the number of letters of ciphertext available. Where PMX started

to fail to work, alternative operators were experimented with. For the three rotor machine and 3380 letters of ciphertext, the level when the GA did not always find the correct solution, a comparison of the various operators performance is presented in section 4. We also present an attempt to combine the operators to try and solve a four rotor machine.

We allowed the mutation operators to move the first wiring of all rotors, and then realligned the mutated offspring so that $\rho'_j(0) = 0, 0 < j < m$. The reason for this was that there was a tendency for the GA to converge to the wrong last rotor. For example, if by fixing the first wiring of the last rotor we are looking for

$$\rho^1_{m-1} = \pi_{m-1} C^{b_{m-1}} \quad \text{where} \quad b_{m-1} = 7$$

then a proposed solution close to a different correct last rotor, for example,

$$\rho^2_{m-1} = \pi_{m-1} C^{b_{m-1}} \quad \text{where} \quad b_{m-1} = 8$$

will have a high fitness. So the GA may converge to a solution for the last rotor with the first and one other wiring incorrect. Thus we allow mutation to move the position of the first wiring with respect to the rest of the permutation. Operators that are not respectful will also move the first wiring on occasion. If an offspring was to mutate, swap or shift mutation was selected randomly. We found that a high probability of mutation tended to speed up convergence to the optimum, and this probability was set at 0.5 for all the results presented in the section 4.

6.2.4 Merge Methods

GAmeter has the following merge methods: best fit, tendency, random, new solutions, replace all of which are described in [40]. We used new solutions exclusively for the results given in section 4, since experimentation indicated it was the most effective at preventing convergence to a local minimum.

6.3 Fitness Measure

Likelihood Function

When the likelihood function is being used as the fitness measure the search space of an m rotor machine is m permutations, and a chromosome, ρ', has fitness $L(\rho'|y)$,

$$L(\rho'|y) = P(y_0|\rho') \times P(y_1|\rho') \times \cdots \times P(y_{n-1}|\rho').$$

The problem with using the likelihood as a fitness measure is that for any reasonably large string size the likelihood is very small indeed, with a huge range of possible values. If roulette selection is used poor solutions can dominate the breeding pool because their fitness is many times greater than the other chromosomes, but still much smaller than the optimum. One way of avoiding this problem is to use ranking selection instead of roulette. This was generally still unsatisfactory because of possible numeric errors and the probability of exceeding the storage capacity of floating point numbers. The solution was to take logs,

$$\log(L(\rho'|y)) = \log(P(y_0|\rho')) + \log(P(y_1|\rho')) + \cdots + \log(P(y_{n-1}|\rho')),$$

and use $-\log(L(\rho'|y))$ as a fitness measure. The objective now is to find the minimum of the negative log likelihood.

Phi Test

The problem with modelling an m rotor system with m permutations is that the building block principle does not really apply. The average fitness of correct short o-schemata within a particular rotor will not necessarily be higher than that of incorrect o-schemata. This dependency means it is highly unlikely that the GA will succesfully search the space of a rotor machine with three or more rotors successfully. A solution to this problem is to reduce the search space to that of the last rotor and use some alternative fitness measure.

The basis for our fitness evaluation is that plaintext enciphered by a monoalphabetic substitution will still exhibit characteristics of the original distribution. A periodic polyalphabetic substitution system with period P can be reduced to a series of monoalphabetic substitution systems by splitting the ciphertext, y, into P text strings, y_j, where

$$y'_j(i) = y'(i * P + j) \text{ for } 0 \leq j < P,\ 0 \leq i < n/P.$$

Each string results from monoalphabetic encipherment. Mapping the ciphertext through the correct last rotor of a rotor machine with periodic rotation will produce a new ciphertext string with the property that the q^{m-1} text strings will exhibit the non-random properties of the plaintext distribution. However, for $m > 3$, this would require a prohibitively large amount of ciphertext. Mapping the ciphertext through the correct last $m-2$ rotors gives q^2 strings.

We can, then, reduce the search space to the possible wirings of the last $m-2$ rotors and consider a fitness measure for the non-randomness exhibited by the q^2 text strings. Various statistics for testing non-randomness were described in chapter 3. The ϕ statisitic is used in these experiments. For a polyalphabetic substitution with period P, and ciphertext y of length n, if we assume n is divisible by P for simplicity, then the P strings of text, y_j, of length n/P can be considered as P observations of random variable ϕ, defined on the sample space of all (n/P) possible letter combinations. Then

$$\bar{\phi} = \frac{\sum_{i=1}^{P} \phi_i}{P}$$

and

$$\sigma_{\bar{\phi}}^2 = \frac{\sigma_{\phi}^2}{P}.$$

By the Central Limit Theorem, [77], the limiting form of the distribution of

$$Z = \frac{\bar{\phi} - \mu_{\phi}}{\sigma_{\bar{\phi}}}$$

is the standard normal distribution (μ_{ϕ} and $\sigma_{\bar{\phi}}$ are defined in chapter 3).

So, if we let $t = q^2$ and $r = \{r_2, \ldots, r_{m-1}\}$ be the displacement functions of the last $m-2$ rotors, the fitness of a proposed last $m-2$ rotors, $\rho'' = \{\rho'_2, \ldots, \rho'_{m-1}\}$, is calculated as follows:

- let y' be the transformed ciphertext found by subjecting ciphertext y to the sequence of substitutions

$$S(r(i), \rho'')^{-1}, \quad \text{for } 0 \leq i < n.$$

- Let y'_j be subtexts of y' such that

$$y'_j(i) = y'(i * t + j) \text{ for } 0 \leq j < t,\ 0 \leq i < n/t.$$

- Calculate A, the observed $\bar{\phi}$ value for the subtexts y'_j.

- Let the fitness of ρ'' be
$$\frac{A - \mu_\phi}{\sigma_{\bar{\phi}}}.$$

The fact we have so many strings means that $\bar{\phi}$ will be approximately normally distributed with small variance. Our estimate of the last $m - 2$ rotors, $\hat{\rho}$, is the set of permutations which maximise the probability of observing a $\bar{\phi}$ value at least as large as A, conditional on the fact that $\mu_{\bar{\phi}} = 0.068944(n/t)(n/t - 1)$ and $\sigma^2_\phi = 0.004344(n/t)^3 + 0.110448(n/t)^2 - 0.114794(n/t)$.

In effect we are using the critical level of the test

$H_0 : \mu_\phi = 0.068944(n/t)(n/t - 1)$ vs
$H_1 : \mu_\phi < 0.068944(n/t)(n/t - 1)$

as the fitness of a proposed set of rotors, assuming the variance is σ^2_ϕ. The assumption of equal variance is not strictly correct for the actual situation because if the strings have not been monoalphabetically enciphered the variance of the ϕ statistic will be different. This does not particularly matter however, since the transformation to the standard normal distribution serves only to standardise the fitness value for different text sizes, and the distribution assuming the alternative hypothesis is true does not effect the fitness value. Alternatively, the fitness could be based on the probability of observing a ϕ statistic as small as actually observed assuming the text is random, in which case the optimisation problem becomes a minimisation problem.

As described above, the phi fitness measure can take negative values. this is a problem if roulette selection is used, as this method assumes positive fitness. One solution is to convert the problem into a minimisation problem by multiplying the fitnesses by -1. For minimisation problems GAmeter uses the deviation from the maximum as the relative fitness, and so the sign of the fitness values is not important. Alternatively the fitness values can be transformed by adding a constant with an absolute value bigger than that of the solution with the smallest fitness, but this seems an arbitrary solution which will inevitably weight the probabilities of selection.

The problem is that the ability of the GA to solve the problem is limited by the fitness measure. With three rotors and 2704 letters of ciphertext each text string has only four letters and, on occasions, the GA found solutions for the three rotor problem with better fitness than the correct rotor. We took 3380 letters as the lower bound for the technique.

6.4 Results

We tested the methods using the rotors VI to VIII and Beta of the German Naval Enigma machine (given in Appendix B). Of the four sets of plaintext used, plaintexts 1 and 2 were generated by random sampling of the single letter distribution, plaintext 3 was taken from an article in a newspaper and plaintext 4 was found by randomly sampling an intentionally skewed distribution. For the last distribution the probability of observing an E or a T was decreased by 0.02 and the probability of observing an X or a Z was increased by 0.02. Plaintext 4 was generated to see how the GA performed with text

that did not strictly match the assumed distribution. The four plaintexts are described in Appendix C.

6.4.1 Cryptanalysis of a Two Rotor Machine

Rotors π_0 and π_1 were used in all experimients. The results below are for the likelihood GA using roulette selection, PMX and new solutions merge method, and give the average figures for solving a two rotor problem with varing amounts of ciphertext. In tables 6.1 and 6.3 the results given for each plaintext are averaged over five runs. In table 6.2 the results are also averaged over five runs except for plaintext 4, where the GA failed to find the correct rotors on two of the five attempts.

n=1500 p=100	Average number of generations	Average number of evaluations	Average time in seconds
plaintext 1	1387	70737	235
plaintext 2	1521	76506	252
plaintext 3	1434	71556	239
plaintext 4	1885	94816	313

Table 6.1: Results for cracking a two rotor machine with 1500 letters of ciphertext and a population, p, of 100

The longest time taken to solve the two rotor problem with 1500 letters of ciphertext was 600 seconds. The GA peformed slightly worse than the iterative technique based an Baum's algorithm, which cracked the 2 rotor machine for all texts with 1000 letters and for plaintext 1 with 800 letters. The relatively poor performance of the GA is due to the problem of inter-

n=1200 p=100	Average number of generations	Average number of evaluations	Average time in seconds
plaintext 1	1566	78613	254
plaintext 2	1612	80600	262
plaintext 3	1850	91575	298
plaintext 4	2342	117803	383

Table 6.2: Results for cracking a two rotor machine with 1200 letters of ciphertext

n=1000 p=200	Average number of generations	Average number of evaluations	Average time in seconds
plaintext 1	2765	276375	864

Table 6.3: Results for cracking a two rotor machine with 1000 letters of ciphertext

dependence between the two rotors, although it is quite impressive that the method worked with such consistency for larger amounts of text.

6.4.2 Cryptanalysis of a Three Rotor Machine

Rotors π_0, π_1, and π_2 were used in all tests. As expected the GA using the likelihood fitness measure failed to solve the three rotor problem. The GA using the ϕ statistic measure of fitness was more sucessful. It managed to find a correct last rotor for text sizes as small as 3380 letters. Some results, using roulette selection, PMX and new solutions merge method, and a population size of 50, are given below. Tables 6.4, 6.5 and 6.6 present the average number of generations, evaluations and time taken to find the last rotor of a three rotor machine over 5 runs for n letters of ciphertext and a popultaion size of p.

n=4732 p=50	Average number of generations	Average number of evaluations	Average time in seconds
plaintext 1	716	18601	111
plaintext 2	1208	31287	190
plaintext 3	1477	38539	232

Table 6.4: Results for finding the last rotor of a three rotor machine with 4732 letters of ciphertext

n=4056 p=50	Average number of Generations	Average number of evaluations	Average time in seconds
plaintext 1	1447	37863	213
plaintext 2	1526	40683	231
plaintext 3	1582	41042	237

Table 6.5: Results for finding the last rotor of a three rotor machine with 4056 letters of ciphertext

n=4732 p=200	Average number of Generations	Average number of evaluations	Average time in seconds
plaintext 4	3049	304102	1737

Table 6.6: Results for finding the last rotor of a three rotor machine with 4732 letters of ciphertext generated from plaintext 4

For plaintext 1, 2 and 3 the GA found the last rotor on every occasion with 4732 and 4056 letters, and the longest time taken was 435 seconds. Once the last rotor was found the simplest method of solving the resulting 2 rotor problem is to use Baum's maximisation algorithm with alternating iterations between row stochastic and column stochastic constraints described in chapter 3. For 15 runs of plaintext 1 2 and 3 with 3380 letters GA found the correct rotor on only four occasions. However, on each run the GA would usually get a substantial number of wirings correct, and often it would be the same or very similar set of wirings. A large portion of the last rotor can be recovered by running the GA several times and looking at the common subsequences. This could provide a way of generating seeds for further searches, either with a GA or some other technique. The GA performed significantly better than the iterative techique for three rotors which was generally unable to solve a three rotor problem.

A Comparison of Genetic Operators

Since PMX was not completely successful at solving the three rotor problem with 3380 letters, we tested the 6 other operators described in the previous chapter. The GA was run 15 times using each operator, 5 times on plaintext 1 2 and 3. A population size of 200 was used. Table 6.7 gives the number of successful runs, the average number of generations required to find the optimal for those successful runs, and the average number of evaluations per second for each operator.

Alternative Rotation Pattern

Operator	Successful runs	Average number of wirings correct for non optimal runs	Number of evaluations	Evaluations per second
PMX	4	9	230579	304
Cycle	3	9	106950	307
Position	3	7	718534	300
Order 1	2	7	394195	304
Order 2	2	8	1119200	302
EER	1	6	198459	258
Union	0	5	-	282

Table 6.7: Comparative performance of 7 genetic operators in finding the last rotor of a three rotor machine using 3380 letters of ciphertext

It could be claimed that using an odometer like rotation problem makes the rotor machine easier to crack. The technique requires that the rotations for the encipherment of the plaintext letters

$$x(i * t + j) \text{ for } 0 \leq j < t,\ 0 \leq i < n/t$$

should not be the same for all i, otherwise each text string y_j has been monoalpabetically enciphered. This is not a restrictive condition, however, since a rotation pattern which did produce the same rotation for each string would reduce the rotor machine to a two rotor problem. Suppose the rotation patterns follow a known pseudo random pattern. We can still split the text so that the text mapped through the last rotor will have been monoalphabetically enciphered, but the rotations for the last rotor will now be different. The following results are for a rotor machine with odometer like rotation for rotors one and two and pseudo random rotation for rotor three.

n=4056 p=50	Average number of Generations	Average number of evaluations	Average time in seconds
plaintext 1	876	22869	127
plaintext 2	1624	40602	223
plaintext 3	1448	36197	203

Table 6.8: Results for cracking a three rotor machine with 4056 letters of ciphertext and a random rotation pattern for the last rotor

6.4.3 Cryptanalysis of a Four Rotor Machine

To use the technique on a four rotor machine it is necessary for the GA to search the space of the last two rotors. Using coincidences rather than likelihood means that much of the information about the plaintext distribution is not used. With three rotors this loss of information was offset by the improvement of the GA with the new representation. However, if the GA is used to search the space of the last two rotors it is expected that there will be little gain from the representation. So, even though the likelihood method worked for two rotors, it was anticipated that the ϕ method would fail for four rotors because the dependency between the rotors would hamper the GA, and this would not be offset by amount of information provided by the fitness measure.

Experimentation confirmed this theory. It is not feasible to restrict the search to the last rotor and use the phi statistic over q^3 substitutions because of the amount of text required. One possible way forward would be imbed a GA or some other search technique to search for the third rotor within a GA searching the space of the last rotor. This would probably be very slow, as each evaluation would require the execution of the embedded procedure.

If rotation is odometer like then the fourth rotor will be stationary for long periods. Since the third rotor only moves every 676 letters, the combined substitution of the last two rotors will only change every 676 letters. It is possible, then, to search the space of these combined substitutions for the solutions $r_1, r_2, \ldots, r_k$, where $k = n/q^2$ and

$$\begin{array}{rcc} r_1 & = & \pi_2 \circ \pi_3 \\ r_2 & = & C^{-1} \circ \pi_2 \circ C^1 \circ \pi_3 \\ \vdots & & \vdots \\ r_k & = & C^{-(k-1)} \circ \pi_2 \circ C^{k-1} \circ \pi_3 \end{array} .$$

Unfortunately searching these k permutations independently gives solutions which are fitter than the optimum. This is because of the relaxation of the constraints on the possible permutation. Further constraints can be introduced because of the fixing of the first wire, for example, if $\pi_2(0) = \pi_3(0) = 0$, then $r_1(0) = 0$ and $r_2(1) \neq 0$, but these are not enough to force the permutations towards the correct solution.

6.4.4 Cryptanalysis of an Enigma Machine

Because of the reflector the phi statistic cannot be used to attack an Enigma machine. One possible way of attacking the Enigma (and a standard rotor machine) is to use the fact that the reflector is stationary and the third rotor is stationary for large periods to search the space of the first two rotors. The reflecting substitution is denoted by M and the plugboard permutation by P. $M = M^{-1}$ and $P = P^{-1}$ because the reflector and the plugboard consist

of transpositions. Let

$$S(R(i),\Pi) = C^{-r_{m-1}(i)}\pi_{m-1}C^{r_{m-1}(i)}C^{-r_{m-2}(i)}\pi_{m-2}C^{r_{m-2}(i)}\cdots C^{-r_0(i)}\pi_0 C^{r_0(i)},$$

then the Enigma enciphering substitution for the i^{th} letter, $E(R(i),\Pi)$, is

$$E(R(i),\Pi) = PS(R(i),\Pi)^{-1}MS(R(i),\Pi)P.$$

The deciphering substitution is identical. Suppose we are attempting to crack a three rotor enigma with no plug board and odometer like rotation.

If we construct a GA to search the space of the first two rotors, then, for any proposed solution, ρ_0', ρ_1', we wish to find the sequence of permutations,

$$< C^{-r_2(i)}(\rho_2')^{-1}C^{r_2(i)}RC^{r_2(i)}\rho_2 C^{-r_2(i)} | 0 \leq i < \infty >,$$

which maximises the likelihood of having observed ciphertext, y. There are at most 26 different permutations in this sequence which change every 676 letters if the rotation is odometer like, and a procedure embedded in the evaluation could conceivably find the optimum permutations for any proposed first two rotors.

Chapter 7

Conclusions

We have shown that a GA can be used to solve a fairly hard cryptographic system using a very simple model of a language. It is a common misconception that the cryptanalysis of rotor machines is a problem easily solved. While it is true that the allies cracked the German Enigma during the war, they had the advantage of having a copy of the machine and knowledge of the enemies operating procedure. Without a copy of the rotors the problem gets much harder. Very little work has been done on the cryptanalysis of rotor machines, mainly because, although there is a company selling cryptographic packages based on an extended Enigma machine, they are generally unsecure if the enemy has a copy of the machine and thus of little practical interest. Of the two papers we could find published in the last twenty years on the cryptanalysis of rotor machines, the GA outperformed Andelman's technique, while Wichmann's [80] attack is with given or chosen plaintext. Indeed, Wichmann states

" We suppose it is very difficult to break the cipher [a modified rotor

machine] under these conditions [ciphertext only]. The easiest way we found was to make a statistical analysis over M^{n+1} [where M is the alphabet size and n the number of rotors] different elements."

By using more complex models, for example models based on observed two letter frequencies or utilising Markov chains, and more complex measures of fitness, possibly based on the test statistics for language recognition described in [26], we would expect an improvement in the performance of the GA.

Little work has been done on the use of genetic algorithms in cryptography. The most important reason for this is that the problems on which a GA are said to perform best are ones where Goldberg's "building block" principle applies, of which he writes in [27]

"Effective processing by genetic algorithms occurs when *building blocks*- relatively short, low order schemata with above average fitness values- combine to form optima or near optima."

But one of the fundamental design principles of a cryptographic system is that the dependency between the key and the ciphertext is over a large enough alphabet to preclude exhaustive search, i.e. that the schemata with above average fitness are of high order. So to use a genetic algorithm on a cryptographic system requires a transformation of the keyspace so that there is some dependency between short key blocks and ciphertext. In chapter 6 we showed that reducing the keyspace of a three rotor machine to the last rotor and using a different fitness measure greatly improved the performance of the GA. A correct short o-schema for last rotor now have above average fitness, whereas the equivalent information for the three rotor will require a

much longer *o*-schema. For example, suppose an *o*-schema with two wirings in the correct position produces 6 coincidences. In the three rotor machine this *o*-schema would not necessarily have above average fitness, and the same information may well require a schema with 6 wirings correct in rotor two and 6 correct in rotor one. So, although using coincidences instead of likelihood means some loss of information, this is more than compensated for by the improved probability of advantageous recombination.

Our experiments with the cryptanalysis of a four rotor machine using the GA to search the space of the last two rotors were largely unsuccessful. The reason for this is again probably interdependence of the rotors. Experimentation with alternative crossover operators may produce better results, but it is thought that unless the problem can be reformulated to one where the building block principle is more applicable, success will be stricly limited.

It is expected that a GA would not perform well searching the keyspace of block ciphers such as DES. This is because of the lack of dependence between subsequences of key/plaintext and ciphertext discussed in chapter 4. There is no obvious fitness measure for which short correct schemata will have above average fitness. However, a research topic might be to examine the use of GAs in conjunction with differential analysis to attack DES. With regard to public key systems, Spillman [75] has used a GA to crack a greatly simplified knapsack cipher by solving the hard knapsack problem, although his knapsack contained only 15 items of about 16 bits each, and a real implementation of a knapsack system would have at least 250 items of 200 to 300 bits each. Besides, all variants of the knapsack except the one proposed by Chor and Rivest [12] have been cracked by other methods, and even if

the only option was to attack the hard knapsack problem, there are probably better methods than a GA(see [55]). As for the systems based on number theory, the problem is finding a fitness measure. RSA relies on the difficulty of factoring large numbers, and it is very hard to imagine a fitness measure of any use to a GA.

From the cryptography side, this thesis has shown that, if a system has a weakness, GAs can be a useful tool in exploiting that weakness. With respect to genetic algorithms, the importance of the representation and fitness measure in the implementation have been the most crucial points raised.

Appendix A

Single Letter Probabilities

letter	probability	letter	probability
A	0.0856	N	0.0707
B	0.0139	O	0.0797
C	0.0279	P	0.0199
D	0.0378	Q	0.0012
E	0.1304	R	0.0677
F	0.0289	S	0.0607
G	0.0199	T	0.1045
H	0.0528	U	0.0249
I	0.0627	V	0.0092
J	0.0013	W	0.0149
K	0.0042	X	0.0017
L	0.0339	Y	0.0199
M	0.0249	Z	0.0008

Table A.1: Single letter probabilities based on the observation of frequencies in the English language

Appendix B

Rotors VI, VII, VIII and Beta of the German Naval Enigma

π_0= {9,15,6,21,14,20,12,5,24,16,1,4,13,7,25,17,3,10,0,18,23,11,8,2,19,22},

π_1= {13,25,9,7,6,17,2,23,12,24,18,22,1,14,20,5,0,8,21,11,15,4,10,16,3,19},

π_2= {5,10,16,7,19,11,23,14,2,1,9,18,15,3,25,17,0,12,4,22,13,8,20,24,6,21},

π_3= {11,4,24,9,21,2,13,8,23,22,15,1,16,12,3,17,19,0,10,25,6,5,20,7,14,18}.

Appendix C

Statisitics Concerning Plaintexts Used in Tests

n	text 1	text 2	text 3	text 4	expected ϕ
3380	1.43491	1.43491	1.36390	1.24556	1.37888
4056	2.20118	2.14793	2.07988	1.86390	2.06832
4732	3.12426	2.89349	2.90532	2.64201	2.89565

Table C.1: Observed and expected mean phi values

letter	expected	text 1	text 2	text 3	text 4
0	684	702	655	672	696
1	111	102	80	145	98
2	223	220	252	206	225
3	302	311	334	300	274
4	1043	1026	1066	971	896
5	223	236	228	155	219
6	159	120	177	172	162
7	418	426	397	471	426
8	501	488	502	560	496
9	10	7	6	18	7
10	33	35	41	57	36
11	271	269	245	377	251
12	199	208	217	182	187
13	565	566	595	526	614
14	637	705	635	627	687
15	159	175	160	168	162
16	9	2	5	3	14
17	541	534	566	531	574
18	485	474	473	504	480
19	836	854	821	755	650
20	199	180	187	211	190
21	73	74	76	60	61
22	119	97	122	175	106
23	13	16	12	18	172
24	159	170	141	132	165
25	6	3	7	4	152

Table C.2: Expected and observed frequencies of plaintext letters

Bibliography

[1] D. Andelman and J. Reeds. On the cryptanalysis of rotor machines and substitution-permutation networks. *IEEE Transactions on Information Theory*, IT-28(4), 1982.

[2] D. Bahler and J. King. An implementation of probabilistic relaxation in the cryptanalysis of simple substitution systems. *Cryptologia*, 16(2), 1992.

[3] J. E. Baker. Adaptive selection methods for genetic algorithms. In Grefenstette [33].

[4] J. E. Baker. Reducing the bias and inefficiency in the selection algorithm. In Grefenstette [34].

[5] V. Barnett. *Comparative Statistical Inference*. John Wiley & Sons, 1985.

[6] L.E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique in the statistical analysis of probabilistic functions of markov chains. *Annals of Mathematical Statistics*, 41(1), 1970.

[7] R. K. Belew and L. B. Booker, editors. *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kauffmann, 1991.

[8] E. Biham and A. Shamir. Differential cryptanalysis of DES-like cryptosystems. *Journal of Cryptology*, 4(1), 1991.

[9] E. F. Brickell. The cryptanalysis of knapsack cryptosystems. In R. D. Ringeisen and F. S. Roberts, editors, *Applications of Discrete Mathematics*. SIAM, 1988.

[10] L. Brown, J. Pieprzyk, and J. Seberry. LOKI - a cryptographic primitive for authentication and secrecy applications. In *Advances in Cryptography - AUSCRYPT'90*. Springer-Verlag, 1990.

[11] J. Carrol and S. Martin. The automated cryptanalysis of substitution ciphers. *Cryptologia*, 10(4), 1986.

[12] B. Chor and R. L. Rivest. A knapsack type public key cryptosystem based on arithmetic in finite fields. In G. R. Blakley and D. Chaum, editors, *Advances in Cryptology - Proceedings of CRYPTO'84*. Springer-Verlag, 1985.

[13] L. J. Cohen. *An Introduction to the Philosophy of Induction and Probability*. Clarendon, 1989.

[14] I. Damgård and Ivan Bjerre. A design principle for hash functions. In G. Brassard, editor, *CRYPTO'89*, pages 416–427. Springer, 1990. Lecture Notes in Computer Science No. 435.

[15] L. Davis. Applying adaptive algorithms to epistatic domains. In *Proceedings of International Conference on Artificial Intelligence*, 1985.

[16] B. De Finetti. *Probability, Induction and Statistics: The Art of Guessing.* John Wiley & Sons, 1972.

[17] A. Deavours and L. Kruh. *Machine Cryptography and Modern Cryptanalysis.* Artech House, 1985.

[18] K. A. DeJong. *An analysis of the behaviour of a class of genetic adaptive systems.* PhD thesis, University of Michigan, 1975.

[19] Y. Desmedt. What happened with knapsack cryptographic schemes. *Performance Limits in Communication, Theory and Practice*, 142, 1988.

[20] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6), 1976.

[21] T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, IT-31(4), 1985.

[22] R. Spillman et al. Use of a genetic algorithm in the cryptanalysis of simple substitution ciphers. *Cryptologia*, 17(1), 1993.

[23] H. Fang, P. Cross, and D. Corne. A promising genetic algorithm approach to job-shop scheduling, rescheduling and open-shop scheduling problems. In Forrest [24].

[24] S. Forrest, editor. *Proceedings of the Fifth International Conference on Genetic Algorithms.* Morgan Kauffmann, 1993.

[25] B. R. Fox and M. B. McMahon. Genetic operators for sequencing problems. In Gregory J. E. Rawlins, editor, *Foundations of Genetic Algorithms*. Morgan Kaufmann, 1991.

[26] R. Ganesan and A. Sherman. Statistical techniques for language recognition: An empirical study using real an simulated english. *Cryptologia*, 18(4), 1994.

[27] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[28] D. Goldberg and R. Lingle. Alleles, loci, and the Travelling Salesman Problem. In Grefenstette [33].

[29] S. Goldwasser and S. Macali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *Proceedings of the 14th ACM Symposium on the Theory of Computing*, 1982.

[30] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer System Sciences*, 28(5), 1984.

[31] S. W. Golomb. *Shift Register Sequences*. Aegean Park Press, 1982.

[32] J. Grefenstette, R. Gopal, B. Rosmaita, and D. Van Gucht. Genetic algorithms for the Travelling Salesman Problem. In Grefenstette [33].

[33] J. J. Grefenstette, editor. *Proceedings of the International Conference on Genetic Algorithms and their Applications*. Lawrence Earlbaum Associates, 1985.

[34] J. J. Grefenstette, editor. *Proceedings of the Second International Conference on Genetic Algorithms.* Lawrence Earlbaum Associates, 1987.

[35] A. Hodges. *Alan Turing: The Enigma.* Vintage, 1983.

[36] J. H. Holland. *Adaption in Natural and Artificial Systems.* the University of Michigan Press, 1975.

[37] T. Jakobsen. A fast method for cryptanalysis of substitution ciphers. *Cryptologia*, 19(3), 1995.

[38] D. Kahn. *The Codebreakers.* Sphere Books Limited, 1973.

[39] A. Kapsalis, V.J. Rayward-Smith, and G.D. Smith. Fast sequential and parallel implementation of genetic algorithms using the GAmeter toolkit. In R.F. Albrecht, C.R. Reeves, and N.C. Steele, editors, *Proceedings of the Int. Conf. on Artificial Neural Nets and Genetic Algorithms.* Springer Verlang, 1993.

[40] A. Kapsalis and G.D. Smith. The GAmeter toolkit manual. School of information systems, computing science technical report sys-c92-01, University of East Anglia, 1992.

[41] J. C. King. An algorithm for the complete automated cryptanalysis of periodic polyalphabetic substitution ciphers. *Cryptologia*, 18(4), 1994.

[42] A. Klapper and M. Goresky. 2-adic shift registers. In *Fast Software Encryption, Cambridge Security Workshop Proceedings.* Springer-Verlag, 1994.

[43] D. Knuth. *The Art of Computer Programming: Volume 2, Seminumerical Algorithms*. Addison-Wesley, 1981.

[44] D. Knuth. Deciphering a linear congruential encryption. *IEEE Transactions on Information Theory*, IT-31(1), 1985.

[45] Alan G. Konheim. *Cryptography:A Primer*. John Wiley & Sons, 1981.

[46] S. Kullback. *Statistical Methods in Cryptanalysis*. Aegean Park Press, 1976.

[47] J. R. Kyburg and H. E. Smokler. *Studies in Subjective Probability*. John Wiley & Sons, 1964.

[48] X. Lai. *On the Design and Securit of Block Ciphers*. Hartung-Goree Verlag Konstanz, 1992.

[49] X. Lai and J. L. Massey. A proposal for a new block encryption standard. In *Advances in Cryptology - EUROCRYPT'90*. Springer-Verlag, 1991.

[50] P. L'Ecuyer. Random numbers for simulation. *Communications of the ACM*, 33(10), 1990.

[51] R. Lidl and H. Niederreiter. *Introduction to Finite Fields and Their Applications*. Cambridge University Press, 1986.

[52] C. H. Lin, C. C. Chang, and R. C. T. Lee. A new public-key cipher system based upon the diophantine equations. *IEEE Transactions on Computers*, 44(1), 1995.

[53] D. V. Lindley. *Introduction to Probability and Statistics from a Bayesian Viewpoint.* Cambridge University Press, 1965.

[54] J. W. Mann, A. Kapsalis, and G. D. Smith. The GAmeter toolkit. In V. J. Rayward-Smith, editor, *Applications of Modern Heuristic Methods.* Alfred Waller, 1995.

[55] S. Martello and P. Toth. *Knapsack Problems.* John Wiley & Sons, 1990.

[56] R.A.J. Mathews. The use of genetic algorithms in cryptanalysis. *Cryptologia*, 17(4), 1993.

[57] H. Muhlenbein. Parallel genetic algorithms, population genetics and combinatorial optimization. In Schaffer [69].

[58] National Bureau of Standards. *Data Encryption Standard.* U.S. Department of Commerce,FIPS Pub 46, 1977.

[59] I. Oliver, D. Smith, and J. Holland. A study of permutation crossover operators on the travelling salesman problem. In Grefenstette [34].

[60] K. Pearson. On a criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can reasonably supposed to have arisen from random sampling. *Phil. Mag.*, 50(5), 1900.

[61] S. Peleg and A. Rosenfeld. Breaking substitution ciphers using a relaxation algorithm. *Communications of the ACM*, 22(11), 1979.

[62] P. W. Poon and J. N. Carter. Genetic algorithm crossover operators for ordering applications. *Computers Operations Research*, 22(1), 1995.

[63] S. J. Press. *Bayesian Statistics: Principles, Models and Applications.* Rinehart and Winston, 1989.

[64] Shamir R. Rivest, A and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commumnications of the ACM*, 21(2), 1978.

[65] N. J. Radcliffe. Forma analysis and random respectful recombination. In Belew and Booker [7].

[66] C. R. Reeves. *Modern Heuristic Techniques for Combinatorial Problems.* Blackwell Scientific Publications, 1993.

[67] T. Ritter. The efficient generation of cryptographic confusion sequences. *Cryptologia*, 15(2), 1991.

[68] L. J. Savage. *The Foundations of Statistical Inference: A Discussion.* Methuen, 1962.

[69] J. D. Schaffer, editor. *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kauffmann, 1989.

[70] B. Schneier. *Applied Cryptography.* John Wiley & Sons, 1995.

[71] A. Shamir. A polynomial-time algorithm for breaking the basic merkle-hellman cryptosystem. *IEE Transactions on Information Theory*, IT-30(5), 1984.

[72] C. E. Shannon. Communication theory of secrecy systems. *Bell. System Technical Journal*, 28, 1949.

[73] A. Shimizu and S. Miyaguchi. Fast data encryption algorithm FEAL. In *Advances in Cryptology - EUROCRYPT'87*. Springer-Verlag, 1988.

[74] G. D. Smith. Economic applications of genetic algorithms. In V. J. Rayward-Smith, editor, *Applications of Modern Heuristic Methods*, pages 71–90. Alfred Waller, 1995.

[75] R. Spillman. Cryptanalysis of knapsack ciphers using genetic algorithms. *Cryptologia*, 17(1), 1993.

[76] T. Starkweather, S. McDaniel, K. Mathias, and D. Whitely. A comparison of genetic sequencing operators. In Belew and Booker [7].

[77] A. Stuart and J. K. Ord. *Kendall's Advanced Theory of Statistics*. Edward Arnold, 1991.

[78] G. Syswerda. Schedule optimization using genetic algorithms. In L. Davis, editor, *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New, York, 1990.

[79] D. Whitley, T. Starkweather, and D'Ann Fuquay. Scheduling problems and travelling salesman: The genetic edge recombination operator. In Schaffer [69].

[80] P. Wichmann. Cryptanalysis of a modified rotor machine. *Lecture notes in computer science*, 434, 1990.