

STUDY OF NEXT BIT/SYMBOL PREDICTOR ALGORITHMS FOR CRYPTOLOGICAL APPLICATIONS

A DISSERTATION

*Submitted in partial fulfilment of the
requirements for the award of the degree*

of

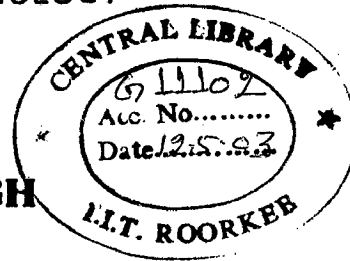
MASTER OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

By

BALBIR SINGH



**ER & DCI
NOIDA**

**IIT Roorkee-ER&DCI, Noida
C-56/1, "Anusandhan Bhawan"**

Sector 62, Noida-201 307

FEBRUARY, 2003

Enrolment No. 019011

~~621.380285~~
SIN

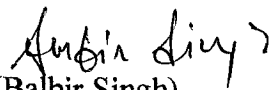
CANDIDATE'S DECLARATION

I hereby declare that the work presented in this dissertation titled "**STUDY OF NEXT BIT/SYMBOL PREDICTOR ALGORITHMS FOR CRYPTOLOGICAL APPLICATIONS**", in partial fulfillment of the requirements for the award of the degree of **Master of Technology in Information Technology**, submitted in **IIT, Roorkee – ER&DCI Campus, Noida**, is an authentic record of my own work carried out during the period from August 2002 to February, 2003 under the guidance of **Mr. N.Rajesh Pillai**, Scientist 'D', D.R.D.O. New Delhi.

I have not submitted the matter embodied in this dissertation for award of any other degree or diploma

Date: 25/2/2003

Place: Noida


(Balbir Singh)

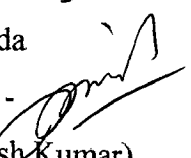
CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.


Date: 25-02-03

Place: Noida

Co-Guide: -


(Mr. Munish Kumar)
Project Engineer,
ER&DCI,
Noida.

Guide: -


(Mr. N. Rajesh Pillai)
Scientist 'D',
Scientific Analysis Group,
Metcalf house D.R.D.O.,
New Delhi.

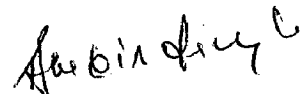
ACKNOWLEDGEMENT

I hereby take the privilege to express my deepest sense of gratitude for **Prof. Prem Vrat**, Director, Indian Institute of Technology, Roorkee, and **Mr. R.K. Verma**, Executive Director, Electronics Research & Development Center of India, Noida for providing me with this valuable opportunity to carry out this work. I am also very grateful to **Prof. A.K. Awasthi**, our Dean, Post Graduate Studies and Research and **Prof. R.P. Agrawal** and **Mr. V.N. Shukla**, our course coordinator for providing the best of the facilities for the completion of this work and constant encouragement towards the goal. I owe special thanks to **Prof. C.E. Veni Madhavan**, Director, Defence Research & Development Organisation, for allowing me to carry out my dissertation at their organization.

I am thankful to my guide **Mr. N.Rajesh Pillai**, Scientist 'D', Scientific Analysis Group, Defense Research & Development Organization, Delhi for his continuous and valuable guidance and providing me with necessary material during the entire course of this work.

I am grateful to **Dr. P.K. Saxena**, Scientist 'G', SAG, DRDO, Delhi for his valuable advice, suggestions and constant encouragement through numerous discussions and demonstrations. My sincere thanks are due to **Dr. S.S. Agrawal**, Emeritus Scientist (CSIR), Central Scientific Instruments Organization, Delhi and **Mr. Munish Kumar**, Project Engineer, ER&DCI, Noida for providing me with their valuable reference for securing the Live project at DRDO, Delhi. I am thankful to **Dr. P.R. Gupta**, Reader, ER&DCI, Noida for her continuous inspiration and support throughout the course of this dissertation.

I extend my gratitude to all of them who have been involved, directly or indirectly, with this work. Thanks to my parents who provided their support and enduring confidence in me during my entire life.



(Balbir Singh)
Enrolment No. 019011

CONTENTS

CANDIDATE'S DECLARATIONS	(i)
ACKNOWLEDGEMENT	(ii)
ABSTRACT	1
1. INTRODUCTION	3
1.1 Objective	3
1.2 Scope	3
1.3 Statement of work done	4
1.4 Organization of dissertation	5
2 LITERATURE SURVEY	7
2.1 Introduction to Cryptography	7
2.2 Random and Pseudo Random Sequences	8
2.2.1 What does random means	8
2.2.2 Definitions and Remarks	9
2.3 Random Bit generation	11
2.3.1 Hardware generation	11
2.3.2 Software generation	12
2.3.3 De-skewing	13
2.4 Machine learning	14
2.4.1 Decision Tree learning	19
2.4.2 Decision Tree Representation	21
2.4.3 How C4.5 works	23
3. DESCRIPTION OF PRNG	25
3.1 Linear congruential generator.	25

3.2 LFSR	25
3.3 RC4	27
3.4 SEAL	28
3.5 BBS	31
4. PREDICTION ALGORITHMS.	33
4.1 A Universal Predictor Based on Pattern Matching	33
4.1.1 Sampled Pattern Matching Predictor	34
4.2 An efficient universal prediction algorithm	35
4.3 Introduction to GNPB Algorithm	35
5. IMPLEMENTATION DETAIL	41
5.1 Next bit/symbol predictor algorithms	41
5.2 Pseudo Random Number Generators	42
6. RESULTS AND DISCUSSION	43
7. CONCLUSION	55
REFERENCES	57

ABSTRACT

The security of many cryptographic systems depends upon the generation of unpredictable quantities. Examples include the secret key in the DES encryption algorithm the primes p , q in the RSA encryption and digital signature schemes, the private key a in the DSA, and the challenges used in challenge-response identification systems[1]. In all these cases, the quantities generated must be of sufficient size and be “random” in the sense that the probability of any particular value being selected must be sufficiently small to preclude an adversary from gaining advantage through optimizing a search strategy based on such probability. For example, the key space for DES has size 2^{56} . If a secret key k were selected using a true random generator, an adversary would on average have to try 2^{55} possible keys before guessing the correct key k . If, on the other hand, a key k were selected by first choosing a 16-bit random secret s , and then expanding it into a 56-bit key k using a complicated but publicly known function f , the adversary would on average only need to try 2^{15} possible keys (obtained by running every possible value for s through the function).

Prediction is important in communication, control, forecasting, investment, molecular biology, security, and other areas. We understand how to do optimal prediction when the data model is known, but there is a need for designing universal prediction algorithms that will perform well no matter what the underlying probabilistic model is. Universal prediction was subject of extensive research over the last 50 years; it dates back to Shannon.

This project is an attempt to predict next bit/symbol of PRNG. One of the predictor algorithm is based on pattern matching. This predictor algorithm is called sampled pattern matching, is a

modification of Ehrenfeucht-Mycielski pseudo random number generator algorithm. It predicts the value of most frequent symbol appearing at so-called sampled positions. The other predictor algorithm is based on machine learning technique and in particular one standard algorithm called C4.5 could be used as an evaluation criteria for PRNG. The last predictor algorithm is A Universal Prediction Algorithm, which yields a prediction error, close to lower bound on the universal prediction error, with limited training data.

The byproduct of this thesis is a tool that that will be used to check the testing resistance of PRNG against next bit/symbol predictor kind of attack .The tool could be used as evaluation criteria of PRNG.

INTRODUCTION

1.1 Objective

A survey of algorithms for next bit/(symbol) predictors from the following viewpoints will be done

- Applicability of next bit/(symbol) predictors for evaluation of existing pseudo-random number generator against next bit/(symbol) predictor type of attacks.
- Testing resistance of some existing pseudo-random number generators against next bit predictor type of attacks.

Pseudo random number generators that will be consider are as following

- Linear Congruential Generators
- Linear Feed Back Shift Register
- RC4
- SEAL

1.2 Scope

This is a research-oriented project. In this project a study and implementation of different algorithms for next bit/(symbol) predictors and pseudo random number generators is done and then next bit/(symbol) predictor algorithms are be applied on to the Pseudo Random Number Generators. One of the next bit/symbol predictor considers the suffixes that are repeating more than twice in the sequence generated by PRNG and predict the symbol/bit that occurs frequently after the suffix. This algorithm is known as SPM, which is published recently in IEEE. The other algorithm is given by Jacob Jiv, and the author claims that it predicts the next/bit symbol even if the length of the sequence considered is less in comparison to the SPM algorithm . This algorithm consider the optimal prediction error for unknown finite-alphabet Markov sources, for prediction algorithms that make inference about the most probable incoming letter, where the distribution of the unknown source is apparent only via a short training sequence of N letters (i.e. N is a polynomial function of K , the order of the Markov source, rather than

the classical case where N is allowed to be exponential in K as in SPM algorithm). A lower bound on the prediction error is formulated for such universal prediction algorithms. A particular universal predictor is introduced, and it is demonstrated that its performance is "optimal" in the sense that it yields a prediction-error, which is close to the lower bound as seen against SPM. This algorithm is published recently in IEEE. The last algorithm is a NESSIE submission. In this work the author presents a variation of the next bit predictor theoretical model that converts it into a classification problem. For solving this classification problem it propose the use of machine learning techniques, and in particular one standard de facto algorithm named C4.5.

As a part of the comparative study, the prediction algorithms will be applied on sequences generated by pseudo-random number generators used in cryptography. The pseudo-random number generators for cryptographic purposes should generate sequences, which are difficult to predict, unless either extremely large amount of sequence is present, or impracticable amount of computational effort is used. We want to study how the newly proposed next bit prediction algorithms mentioned above perform when used against cryptographic pseudo-random number generators. Both basic building blocks of cryptographic Pseudo Random number generators and full industry grade stream ciphers will be studied for resistance to next bit predictor kind of attacks. The building blocks Considered will be linear feedback shift register (LFSR) and Linear Congruential generator.

These systems will be implemented and the sequences generated by them will be studied for resistance against next bit prediction. We will apply next bit prediction algorithms on the sequences generated by the above systems. Sequences generated from the above systems will be tested for applicability of next - bit prediction kind of attacks.

1.3 Statement of the work done

The following algorithms for next bit prediction will be studied and implemented.

1. A universal prediction algorithm[2]. Given by Jacob Ziv, . This algorithm is published recently in IEEE transactions of information theory.
2. Sampled pattern matching (SPM)[3]. This algorithm is a modification of Ehrenfeucht-Mycielski pseudo random generator algorithm, and this work is done by philipe jacquet,wojciech szpankowski and Izydor Apostol. This work is published in IEEE , transactions of information theory recently.
3. General next bit prediction algorithm[4] this algorithm is a NESSIE (new European schemes for signature integrity and encryption) submission, published recently.

Pseudo random number generators that will implemented are as following

1. Linear Congruential Generators
2. Linear Feed Back Shift Register
3. RC4
4. SEAL

Then apply next bit/symbol predictor on the pseudo random number generators to predict the next bit/symbol and address the following issues.

- Find the relation/formula for the minimum length of sequence required for the next bit predictor to work (predict next bit correctly with a probability better than random guessing). These relations would depend on both the predictor algorithm and the sequence.
- What are the typical values (for minimal length) the generators being considered?
- Applicability of next bit/(symbol) predictors for evaluation of existing pseudo-random number generator against next bit/(symbol) predictor type of attacks on cryptographic pseudo-random number generators.
- Testing resistance of some existing pseudo-random number generators against next bit predictor type of attacks.

1.4 Organization of Dissertation

- Chapter 2 describes the basic properties of pseudo random number generators, different kind of PRNG i.e. hardware ,s/w. It also gives an introduction to machine learning and c4.5.

- Chapter 3 describes various PRNG that are considered for implementation and testing resistance against next bit/symbol kind of attack.
- Chapter 4 describes various next bit/symbol predictors that are studied and implemented .
- Chapter 5 describes the implementation details of PRNGs and prediction algorithms.
- Chapter 6 describes the results and discussions.
- Chapter 6 concludes the thesis and describes the scope for the future work.

LITERATURE SURVEY

2.1 Cryptography[2]

- The art or science encompassing the principles and methods of transforming an intelligible message into one that is unintelligible, and then retransforming that message back to its original form.
- It is a Science which deals with keeping information secure, transmitting it over channel and receiving at other.

Encryption

The process of transforming intelligible text into unintelligible form is called encryption. The encrypted text is called cipher text

Decryption

The process of transforming cipher text back to plain text is called decryption

Public key cryptosystem:

Public key crypto system relay on one key for encryption and a different and related key for decryption. These algorithms have following characteristics.

- 1) It is computationally infeasible to find the decryption key if given only the Cryptographical algorithm and the encryption key.
- 2) Either of the two related keys can be used for encryption with the other been used for decryption.

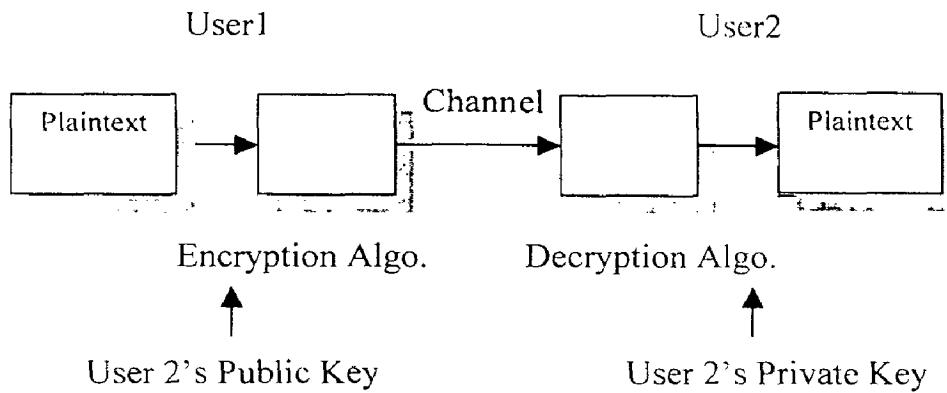


Figure 2.1: Encryption

Figure 2.1 depicting encryption provides secrecy only. Figure 2.2 shows a system which provides authentication as only a specific known public key can be used to decrypt the cipher text . thus the identity of the sender is confirmed or validated.

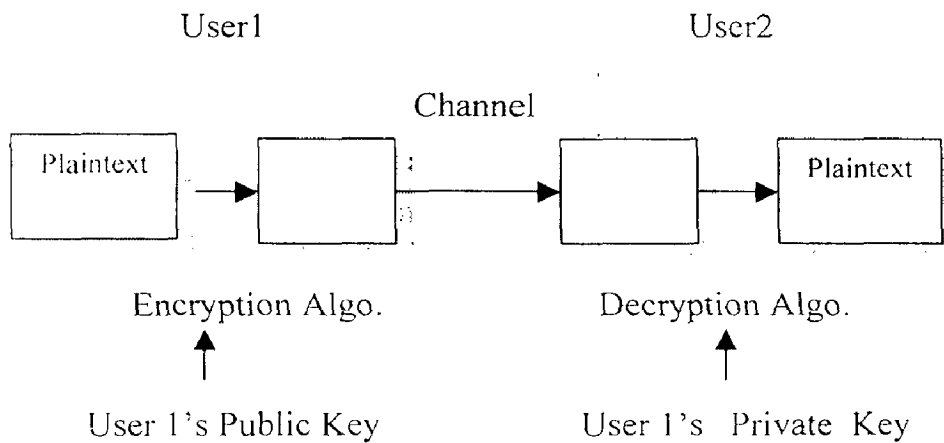


Figure 2.2 : Authentication

2.2 Random Sequence and Pseudo Random Sequence

2.2.1 What does random mean:

The three **Uns**[1].

A true random number generator has three important properties:

Its Unbiased. All values of whatever sample size is collected are equiprobable.

Its Unpredictable. It is impossible to predict what the next output will be, given all the previous outputs, but not the internal "hidden" state.

Its Unreproducible. Two of the same generators, given the same starting conditions, will produce different outputs.

Usually when a person says they have a "good" random number generator, they mean it is unbiased. If they say they have a "true" RNG, they usually mean it's Unreproducible. If they say it's "cryptographically strong" they mean it's unpredictable. Very rarely do they mean it's all three Uns. This isn't necessarily a bad thing, but it's worth remembering when evaluating claims that one RNG is "better" than another. A sequence generator is real random if it has this additional third property. It cannot be reliably reproduced. if you run the sequence generator twice with the exact same input(at least as exact as humanly possible),you will get two completely unrelated random sequences.

Pseudo Random Sequence[5]

A pseudo random sequence is one that looks random .The sequence period should be long enough so that finite sequence of reasonable length – that is, one that is actually used – is not periodic.

A sequence generator is a pseudo – random if it has the following property.

- It looks random. This means that it passes all the statistical tests of randomness that we can find.

For a system to be cryptograph ally secure pseudo-random, it must also have this property

- It is unpredictable. It must be computationally infeasible to predict what the next random bit will be, given complete knowledge of the algorithm or hardware generating the sequence and all of the previous bits in stream.

2.2.2 Definitions and Remark[1]

Definition A random bit generator is a device or algorithm which outputs a sequence of statistically independent and unbiased binary digits.

Remark (random bits vs. random numbers) A random bit generator can be used to generate (uniformly distributed) random numbers. For example, a random integer in the interval $[0, n]$ can be obtained by generating a random bit sequence of length $\lceil \lg n \rceil + 1$, and converting it to an integer; if the resulting integer exceeds n , one option is to discard it and generate a new random bit. Ideally, secrets required in cryptographic algorithms and protocols should be generated with a (true) random bit generator. However, the generation of random bits is an inefficient procedure in most practical environments. Moreover, it may be impractical to securely store and transmit a large number of random bits.

Definition A pseudo random bit generator (PRBG) is a deterministic algorithm which, given a truly random binary sequence of length k , outputs a binary sequence of length $l \gg k$ which “appears” to be random. The input to the PRBG is called the seed, while the output of the PRBG is called a pseudo random bit sequence.

The output of a PRBG is *not* random; in fact, the number of possible output sequences is at most a small fraction, namely $(2^k / 2^l)$, of all possible binary sequences of length l . The intent is to take a small truly random sequence and expand it to a sequence of much larger length such a way that an adversary cannot efficiently distinguish between output sequences of the PRBG and truly random sequences of length l . [1] discusses ad-hoc techniques for pseudo random bit generation. In order to gain confidence that such generators are secure, they should be subjected to a variety of statistical tests designed to detect the specific characteristics expected of random sequences. As the example of PRNG demonstrates, passing these statistical tests is a necessary but not sufficient condition for a generator to be secure. A minimum-security requirement for a pseudo random bit generator is that the length k of the random seed should be sufficiently large so that a search over 2^k elements (the total number of possible seeds) is infeasible for the adversary. Two general requirements are that the output sequences of a PRBG should be statistically indistinguishable from truly random sequences, and the output bits should be unpredictable to an adversary with limited computational resources

Definition A pseudo random bit generator is said to pass all polynomial-time² statistical tests if no polynomial-time algorithm can correctly distinguish between an output sequence of the generator and a truly random sequence of the same length with probability significantly greater than $1/2$.

Definition A pseudo random bit generator is said to pass the next-bit test if there is no polynomial-time algorithm which, on input of the first L bits of an output sequence s , can predict the $(L + 1)$ st bit of s with probability significantly greater than $1/2$. Although the above definition appears to impose a more stringent security requirement on pseudo random bit generators than the above Definition does, the next result asserts that they are, in fact, equivalent.

Fact (universality of the next-bit test) A pseudo random bit generator passes the next-bit test if and only if it passes all polynomial-time statistical tests.

Definition A PRBG that passes the next-bit test (possibly under some plausible but unproved mathematical assumption such as the intractability of factoring integers) is called a cryptographically secure pseudo random bit generator (CSPRBG)[1].

2.3 Random bit generation

A true random bit generator requires a naturally occurring source of randomness. Designing a hardware device or software program to exploit this randomness and produce a bit sequence that is free of biases and correlations is a difficult task. Additionally, for most cryptographic applications, the generator must not be subject to observation or manipulation by an adversary. This section surveys some potential sources of random bits. Random bit generators based on natural sources of randomness are subject to influence by external factors, and also to malfunction. It is imperative that such devices be tested periodically[1].

2.3.1 Hardware-based generators

Hardware-based random bit generators exploit the randomness which occurs in some physical phenomena. Such physical processes may produce bits that are biased or correlated, in which case they should be subjected to de-skewing techniques mentioned in (iii) below.

Examples of such physical phenomena include:

1. Elapsed time between emission of particles during radioactive decay;
2. Thermal noise from a semiconductor diode or resistor;
3. The frequency instability of a free running oscillator;
4. The amount a metal insulator semiconductor capacitor is charged during a fixed period of time;
5. Air turbulence within a sealed disk drive which causes random fluctuations in disk drive sector read latency times; and
6. sound from a microphone or video input from a camera.

Generators based on the first two phenomena would, in general, have to be built externally to the device using the random bits, and hence may be subject to observation or manipulation by an adversary. Generators based on oscillators and capacitors can be built on VLSI devices; they can be enclosed in tamper-resistant hardware, and hence shielded from active adversaries.

2.3.2 Software-based generators

Designing a random bit generator in software is even more difficult than doing so in hardware.

Processes upon which software random bit generators may be based include:

1. The system clock;
2. Elapsed time between keystrokes or mouse movement;
3. Content of input/output buffers;
4. User input; and
5. Operating system values such as system load and network statistics.

The behavior of such processes can vary considerably depending on various factors, such as the computer platform. It may also be difficult to prevent an adversary from observing or manipulating these processes. For instance, if the adversary has a

rough idea of when a random sequence was generated, she can guess the content of the system clock at that time with a high degree of accuracy. A well-designed software random bit generator should utilize as many good sources of randomness as are available. Using many sources guards against the possibility of a few of the sources failing, or being observed or manipulated by an adversary. Each source should be sampled, and the sampled sequences should be combined using a complex mixing function; one recommended technique for accomplishing this is to apply a cryptographic hash function such as SHA-1 or MD5 to a concatenation of the sampled sequences. The purpose of the mixing function is to distill the (true) random bits from the sampled sequences.

2.3.3 De-skewing

A natural source of random bits may be defective in that the output bits may be biased (the probability of the source emitting a 1 is not equal to $1/2$) or correlated (the probability of the source emitting a 1 depends on previous bits emitted). There are various techniques for generating truly random bit sequences from the output bits of such a defective generator; such techniques are called de-skewing techniques.

2.3.4 Statistical tests

Some tests designed to measure the quality of a generator purported to be a random bit generator. While it is impossible to give a mathematical proof that a generator is indeed a random bit generator, the tests described here help detect certain kinds of weaknesses the generator may have. This is accomplished by taking a sample output sequence of the generator and subjecting it to various statistical tests. Each statistical test determines whether the sequence possesses a certain attribute that a truly random sequence would be likely to exhibit; the conclusion of each test is not definite, but rather probabilistic. An example of such an attribute is that the sequence should have roughly the same number of 0's as 1's. If the sequence is deemed to have failed any one of the statistical tests, the generator may be rejected as being non-random; alternatively, the generator may be subjected to further testing. On the other hand, if the sequence passes all of the statistical tests, the generator is accepted as being random. More precisely, the term "accepted" should be replaced by "not rejected", since passing the

tests merely provides probabilistic evidence that the generator produces sequences which have certain characteristics of random sequences[1] provide some relevant background in statistics.

- ❖ The normal and chi-square distributions
- ❖ Hypothesis testing
- ❖ Golomb's randomness postulates

Application of pseudo-random sequence generator[5]

- a) To select random samples from a larger set
- b) In cryptography
- c) running securities protocol
- d) As initial values in procedures to generate prime numbers
- e) To test computer programs
- f) For fun

2.4 Machine Learning

Machine learning[6] is a process, which causes systems to improve with experience. Machine Learning (ML) is one of the most important areas of Artificial Intelligence (AI), a subject of great interest in which improvements and new developments are constantly made. As its name reveals, its objectives are quite ambitious, that is, to develop algorithms that can learn, adapt its behavior, self-improve, a core characteristic of human intelligence. ML techniques have been successfully applied to a wide range of applications like problem solving, theorem proving, natural language processing, speech recognition, vision pattern searching, robotics, industrial process control, planning, game playing, data mining, and so on.

My approach belongs to an inductive paradigm, so called because we try to generate a concept ("next bit is 0", for example) by generalizing from a set of examples and counter-examples of the concept. The task is to build a concept description that can predict the value of the class for all (or most of them) previously seen instances. Why do we use this inductive paradigm? We believe it is the most natural solution to a problem of

this type, were we try to obtain a discrete pattern recognition, discovering a collection of features that are related with membership in a predefined equivalence class.

Elements of a Learning Task [7]

Representation

1. Items of Experience
 - $i \in I$
2. Space of Available Actions
 - $a \in \Lambda$
3. Evaluation
 - $v(a, i)$
4. Base Performance System
 - $b: I \rightarrow A$
5. Learning System
 - $L: (i_1, a_1, v_1) \dots (i_n, a_n, v_n) \rightarrow b$
 - Maps training experience sequence to base performance system.

Types of Learning Problems

Fundamental Distinctions

- Between problems rather than methods.
- Influences choice of method.

1. Batch versus Online Learning:

- **Batch Learning**
 - Training phase (no responsibility for performance during learning), then testing.
 - Synthesizing a base performance system.
 - Pure exploration.
- **Online Learning**
 - No separate phase.
 - Learning while doing.

- Have to decide between choosing actions to perform well now versus gaining knowledge to perform well later.

2. Learning from Complete versus Partial Feedback:

○ Complete Feedback

- Each item of experience evaluates every action (or base system).
- Tells what best action would have been.

○ Partial Feedback

- Each item of experience evaluates only a subset of possible actions (or base systems).
- Only tells you how you did.
- Creates exploration/exploitation tradeoff in online setting.
- Should the reward be optimized with what is already known or should learning be attempted?
- In batch learning, exploration would be chosen since there is no responsibility for performance.

○ Pointwise Feedback

- Evaluates single action.
- Optimization problem: (Controller, State) → Action.

3. Passive versus Active Learning

- **Passive:** observation.
- **Active:** experimentation, exploitation.

4. Learning in Acausal versus Causal Situations

- **Acausal:** actions do not affect future experience.
 - e.g., rain prediction
- **Causal:** actions do effect future experience.

5. Learning in Stationary versus Nonstationary Environments

- **Stationary:** evaluation doesn't change.
- **Nonstationary:** evaluation changes with time.

Concept Learning

The problem of inducing general functions from specific training examples is central to learning.

Concept learning acquires the definition of a general category given a sample of positive and negative training examples of the category, the method of which is the problem of searching through a hypothesis space for a hypothesis that best fits a given set of training examples.

A **hypothesis space**, in turn, is a predefined space of potential hypotheses, often implicitly defined by the hypothesis representation.

Learning a Function from Examples

- An example of concept learning where the concepts are mathematical functions.
Shown in figure 2.3:
- Domain X: descriptions
- Domain Y: predictions
- H: hypothesis space; the set of all possible hypotheses
- h: target hypothesis

Idea: to extrapolate observed y's over all X.

Hope: to predict well on future y's given x's.

Require: there must be regularities to be found!

(Note type: batch, complete, passive (we are not choosing which x), acausal, stationary).

Many Research Communities

- Representation choice differs.
- Prefer maximum level of abstraction.

Traditional Statistics

- $h: \mathbb{R}^n \rightarrow \mathbb{R}$
- Squared prediction error.
- h is a linear function.

Traditional Pattern Recognition

- $h: \mathbb{R} \rightarrow \{0, 1\}$
- Right/wrong prediction error.
- h is a linear discriminant boundary.

"Symbolic" Machine Learning

- $h: \{\text{attribute-value vectors}\} \rightarrow \{0, 1\}$
- h is a simple Boolean function (e.g. a decision tree).

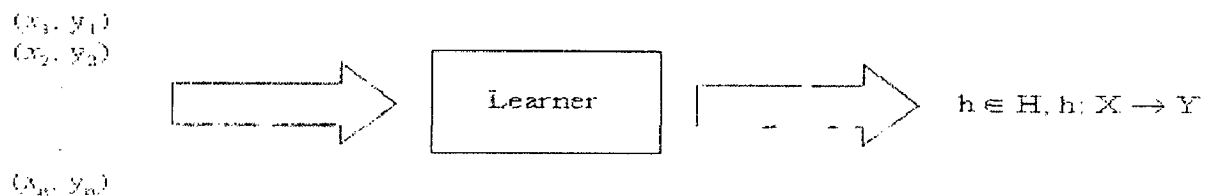


Figure 2.4: Concept Learning

Neural Networks

- $h: \mathbb{R}^n \rightarrow \mathbb{R}$
- h is a feed forward neural net.

Inductive Logic Programming

- $h: \{\text{term structure}\} \rightarrow \{0, 1\}$
- h is a simple logic program.

Learning Theory

- Postulate mathematical model of learning situations.
- Rigorously establish achievable levels of learning performance.
- Characterize/quantify value of prior knowledge and constraints.

- Identify limits/possibilities of learning algorithms.

Standard Learning Algorithm

Given a batch set of training data $S = \{(x_1, y_1) \dots (x_t, y_t)\}$, consider a fixed hypothesis class H and compute a function $h \in H$ that minimizes empirical error.

$$\sum_{j=1}^t \frac{\text{err}(h(x_j, y_j))}{t} = \text{err}(h, S)$$

Example: Least-Squares Linear Regression

Here, the standard learning algorithm corresponds to least squares linear regression.

$$X = \mathbb{R}^n, Y = \mathbb{R}$$

$$\text{err}(y', y) = (y' - y)^2$$

$$H = \{\text{linear functions } \mathbb{R}^n \rightarrow \mathbb{R}\}$$

Choosing a Hypothesis Space

In practice, for a given problem $X \rightarrow Y$, which hypothesis space H do we choose?

$$H_1 \text{ vs. } H_2$$

richer	easier to search
--------	---------------------

Question: since we know the true function $f: X \rightarrow Y$, should we make H as "expressive" as possible and let the training data do the rest?

- e.g., quadratic
- e.g., high degree polynomial
- e.g., complex neural network.

Answer: No

Reason: overfitting!

2.4.1 Decision Tree Learning[8]

Robust to noisy data and capable of learning disjunctive expressions, **decision tree learning**, a method for approximating discrete-valued target functions, is one of the most widely used and practical methods for inductive inference.

Appropriate Problems for Decision Tree Learning

Decision tree learning is generally best suited to problems with the following characteristics:

- Instances are represented by **attribute-value pairs**.
 - Instances are described by a fixed set of attributes (e.g., temperature) and their values (e.g., hot).
 - The easiest situation for decision tree learning occurs when each attribute takes on a small number of disjoint possible values (e.g., hot, mild, cold).
 - Extensions to the basic algorithm allow handling real-valued attributes as well (e.g., a floating point temperature).
- The target function has **discrete output values**.
 - A decision tree assigns a classification to each example.
 - Simplest case exists when there are only two possible classes (**Boolean classification**).
 - Decision tree methods can also be easily extended to learning functions with more than two possible output values.
 - A more substantial extension allows learning target functions with real-valued outputs, although the application of decision trees in this setting is less common.
- Disjunctive descriptions may be required.
 - Decision trees naturally represent disjunctive expressions.
- The training data may contain errors.

- Decision tree learning methods are robust to errors - both errors in classifications of the training examples and errors in the attribute values that describe these examples.
- The training data may contain missing attribute values.
 - Decision tree methods can be used even when some training examples have unknown values (e.g., humidity is known for only a fraction of the examples).

Learned functions are either represented by a decision tree or re-represented as sets of if-then rules to improve readability.

2.4.2 Decision Tree Representation

A **decision tree** is an arrangement of tests that prescribes an appropriate test at every step in an analysis. In general, decision trees represent a disjunction of conjunctions of constraints on the attribute-values of instances. Each path from the tree root to a leaf corresponds to a conjunction of attribute tests, and the tree itself to a disjunction of these conjunctions. More specifically, decision trees classify **instances** by sorting them down the tree from the **root node** to some **leaf node**, which provides the classification of the instance. Each node in the tree specifies a **test** of some **attribute** of the instance, and each **branch** descending from that node corresponds to one of the possible **values** for this attribute. An instance is classified by starting at the root node of the decision tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute. This process is then repeated at the node on this branch and so on until leaf node is reached as shown in figure 2.5.

Diagram explanation

- Each nonleaf node is connected to a test that splits its set of possible answers into subsets corresponding to different test results.
- Each branch carries a particular test result's subset to another node.
- Each node is connected to a set of possible answers.

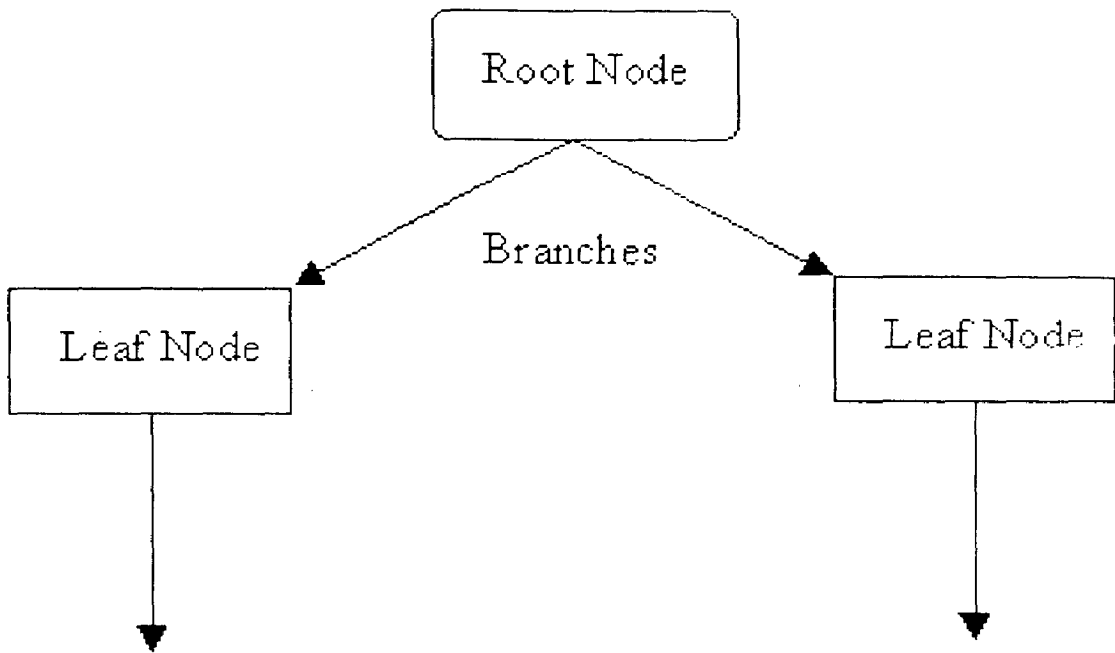


Figure 2.4 : Decision Tree Representation

Occam's Razor (Specialized to Decision Trees)

The smallest decision tree that is consistent with the samples is the one that is most likely to identify unknown objects correctly as shown in figure 2.5.

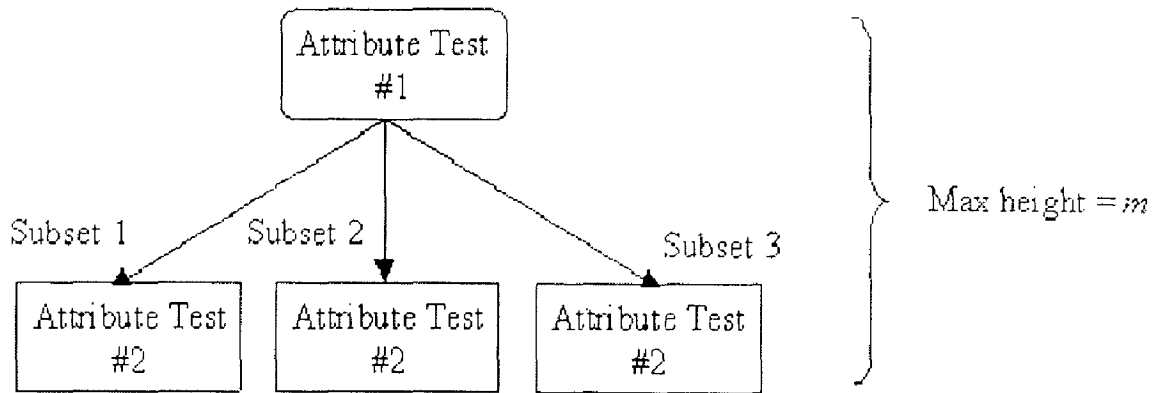


Figure 2.5 : Decision Tree

Given m attributes, a decision tree may have a maximum height of m . Rather than building all the possible trees, measuring the size of each, and choosing the smallest tree that best fits the data, uses Quinlan's ID3 algorithm for constructing a decision tree.[7]

2.4.3 How C4.5 works

General Next Bit Prediction algorithm uses machine learning technique .One of the machine learning technique is implemented in a C algorithm called C4.5 [6].Basically, C4.5 is a successor of ID3 with some improvements and added capabilities.

Suppose we are given a set of instances that are simply a number of attribute/value pairs each of which is assigned a category (or classification). The problem is to determine a decision tree that, on the values of some of the attributes of an instance, could correctly classify it. This is not, generally, an easy task to do as we can and discrete or continuous attributes, unknown attributes values, noise, etc.

What is exactly a decision tree? It can be defined as a tree in which each node is an attribute, each arc from this node is a possible value for that attribute, and each leaf is the expected value for the category of the instance obtained following all the path from the root of the tree to that leaf.

How do we construct decision trees? The basic idea is to decide at each node which of the non-used attributes is most informative for the classification of all the instances represented by the path from the root to that node. Then, applying this idea recursively in every node (see below) the tree is constructed.

And, how can we measure how informative is a node? By means of a gain function which uses entropy in the Shannon sense. This gain function $G(a_i)$ tells us how informative the attribute a_i is for the classification of our actual set of instances, that is, the difference between the information needed to identify an element of class C after and before the value of the attribute a_i is used as a node. This can also be seen as the difference in the entropy of the set of instances after and before we have divided this set depending on the value of attribute a_i . The greater this gain is, the better is our choice of attribute a_i for our classification, and for our decision tree.

DESCRIPTION OF PRNG

3.1 Linear Congruential generator.

LCG is a pseudo sequence generators of the form[5]

$$X_n = ((aX_{n-1} + b) \bmod m), n \geq 1$$

In which X_n is the n th number of the sequence and X_{n-1} is the previous number of the sequence. The variable a , b and m are constants: a is multiplier b is the increment and m is the modulus. The key or seed is the value of X_0 , which is secret. This generators has a period no greater than m . if a , b and m are properly chosen then the generator will be a maximal period generator and having period of m .

3.2 Linear Feed Back Shift Register

Shift register sequences are used in both cryptography and coding theory. There is a wealth of theory about them. stream ciphers based on shift registers have been the workhorse of military cryptography since the beginning of electronics [5].

A feed back shift register is made up of two parts:

1. A shift registers.
2. A feed back function

A **linear feedback shift register** is the building block for many simple stream ciphers. The function of a linear feedback shift register is to produce the key-stream. This example is a 4-stage linear feedback shift register. They are actually much longer.

• 1111	
• 0111	
• 0011	
• 0001	At each tick of the internal clock:
• 1000	• The four bits shift right
• 0100	• The right-most bit is dropped
• 0010	• The last two bits (before the shift) are XORed
• 1001	• The XORed result is placed in left-most position of the register
• 1100	
• 0110	
• 1011	This register sequence represents one "state"
• 0101	The register then iterates
• 1010	
• 1101	
• 1110	
• 1111	

Table 3.1: Operation of Pseudo random number generator($x^4 + x + 1$)

- The initial value of the linear feedback shift register may be part of the crypto-variable (encryption key).
- The initial value should never be: 0000 as subsequent states of the register are then very predictable.
- Other options for the initial value besides 1111 are: 0111, 0011, 1001, 1100, 1110

- Once there is repetition or linearity from one state to another - crypto-analysis becomes easier

Linear Feedback Shift Registers

- Should have long periods without repetition
- Should be statistically unpredictable
- Should have functional complexity
- Should be statistically unbiased (the number of 1s and 0s should be about equal)
- The key-stream should not be linearly related to the crypto-variable (encryption key)

3.3 RC4

RC4 is a variable –key-size stream cipher developed in 1987 by Ron Rivest for rsa data security, inc[1].

RC4 is simple to describe. The algorithm works in OFB:

The key stream is independent of plaintext. It has a 8*8 S-box: S1,S2.....S255.

The entries are a permutation of numbers 0 through 255, and permutation is a function of the variable-length key. It has two counters I and j initialized to zero.

To generate random bytes do the following.

```

i=(i+1) mod 256
j=(j+Si) mod 256
Swap Si and Sj
t=(Si+Sj) mod 256
K=St

```

The byte is XORED with the plaintext to produced cipher text or XORED with the cipher text to produced plaintext. Encryption is fast about 10 times than DES

Initializing the s-box is also easy .first fill it linearly : S₀=0,S₁+1.....S₂₅₅=255

Then fill another 256-byte array with the key, repeating the key as necessary to fill the entire array: K_0, K_1, \dots, K_{255} . Set the index j to zero then:

```

For i=0 to 255
j=(j+Si+Ki) mod 256
Swap Si and Sj

```

3.4 SEAL

Seal is software – efficient stream cipher designed at IBM by phil rogaway and Don coppersmith. The algorithm was optimized for 32 bit processors: to run well it needs eight 32- bit registers and a cache of few kilobytes. Using a relatively slow operation. SEAL preprocesses the key operation into a set of a table. These tables are then used to speed up encryption and decryption.

SEAL (Software-optimized Encryption Algorithm) is a binary additive stream cipher [1] that was proposed in 1993. Since it is relatively new, it has not yet received much scrutiny from the cryptographic community. this stream ciphers was specifically designed for efficient software implementation and, in particular, for 32-bit processors. SEAL is a length-increasing pseudo random function which maps a 32-bit sequence number n to an L -bit keystream under control of a 160-bit secret key a . In the preprocessing stage , the key is stretched into larger tables using the table generation function G_a , this function is based on the Secure Hash Algorithm SHA-1 . Subsequent to this preprocessing, keystream generation requires about 5 machine instructions per byte, and is an order of magnitude faster than DES. The following notation is used in SEAL for 32-bit quantities A, B, C, D, X_i , and Y_j :

–
 \bar{A} : bitwise complement of A
 $A \wedge B, A \vee B, A \oplus B$: bitwise AND, inclusive-OR, exclusive-OR
 “ $A \leftarrow s$ ” : 32-bit result of rotating A -left through s positions
 “ $A \Rightarrow s$ ” : 32-bit result of rotating A right through s positions
 $A + B$: mod 2^{32} sum of the unsigned integers A and B

$f(B,C,D)$ =(defined) $(B \wedge C) \vee (\bar{B} \wedge D)$;
 $g(B,C,D)$ =(defined) $(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$;

$h(B,C,D) = (\text{defined}) B \oplus C \oplus D$

$A \parallel B$: concatenation of A and B

$(X_1, \dots, X_j) (Y_1, \dots, Y_j)$: simultaneous assignments $(X_i \leftarrow Y_i)$, where (Y_1, \dots, Y_j) is evaluated prior to any assignments.

$G_a(i)$

INPUT: a 160-bit string a and an integer i, $0 < i < 2^{32}$

OUTPUT: a 160-bit string, denoted $G_a(i)$.

1. Definition of constants. Define four 32-bit constants (in hex): $y_1 = 0x5a827999$,
 $y_2 = 0x6ed9eba1$, $y_3 = 0x8f1bbcdc$, $y_4 = 0xca62c1d6$.

2. Table-generation function.

(initialize 80 32-bit words X_0, X_1, \dots, X_{79})

Set $X_0 \leftarrow i$. For j from 1 to 15 do: $X_j \leftarrow 0x00000000$.

For j from 16 to 79 do: $X_j \leftarrow (X_{j-3} \oplus X_{j-8} \oplus X_{j-14} \oplus X_{j-16}) \ll 1$.

(initialize working variables)

Break up the 160-bit string a into five 32-bit words: $a = H0 \parallel H1 \parallel H2 \parallel H3 \parallel H4$.

$(A, B, C, D, E) \leftarrow (H0, H1, H2, H3, H4)$.

(execute four rounds of 20 steps, then update; t is a temporary variable)

(Round 1) For j from 0 to 19 do the following:

$t \leftarrow ((A \ll 5) + h(B, C, D) + E + X_j + y_1)$,

$(A, B, C, D, E) \leftarrow (t, A, B \ll 30, C, D)$.

(Round 2) For j from 20 to 39 do the following:

$t \leftarrow ((A \ll 5) + h(B, C, D) + E + X_j + y_2)$,

$(A, B, C, D, E) \leftarrow (t, A, B \ll 30, C, D)$.

(Round 3) For j from 40 to 59 do the following:

$t \leftarrow ((A \ll 5) + g(B, C, D) + E + X_j + y_3)$,

$(A, B, C, D, E) \leftarrow (t, A, B \ll 30; C; D)$.

(Round 4) For j from 60 to 79 do the following:

$t \leftarrow ((A \ll 5) + h(B, C, D) + E + X_j + y_4)$,

$(A, B, C, D, E) \leftarrow (t, A, B \ll 30, C, D)$.

(Update chaining values)

$(H_{10}, H_{11}, H_{12}, H_{13}, H_{14}) \leftarrow (H_{10} \oplus A, H_{11} \oplus B, H_{12} \oplus C, H_{13} \oplus D, H_{14} \oplus E).$

(completion) The value of $G_a(i)$ is the 160-bit string $H_{10} \parallel H_{11} \parallel H_{12} \parallel H_{13} \parallel H_{14}$

Algorithm Keystream generator for SEAL.

SEAL(a,n)

INPUT: a 160-bit string a (the secret key), a (non-secret) integer n, $0 \leq n < 2^{32}$ (the sequence number), and the desired bitlength L of the keystream.

OUTPUT: keystream y of bitlength L0, where L0 is the least multiple of 128.

1. Table generation. Generate the tables T, S, and R, whose entries are 32-bit words.

The function F used below is defined by $F_a(i) = H_{i \bmod 5}^i$, where H_i

$$H_0^i, H_1^i, H_2^i, H_3^i, H_4^i = G_a \lfloor (i/5) \rfloor$$

and where the function G_a is defined in above Algorithm.

1.1 For i from 0 to 511 do the following:

$$T[i] \leftarrow F_a(i)$$

1.2 For j from 0 to 255 do the following:

$$S[j] \leftarrow F_a(0x00010001j).$$

1.3 For k from 0 $\lfloor (L-1)/8192 \rfloor - 1$

Do :

$$R[k] \leftarrow F_a(0x00010001k).$$

2. Initialization procedure. The following is a description of the subroutine Initialize(n, l, A, B, C, D, n1, n2, n3, n4) which takes as input a 32-bitword n and an integer l, and puts eight 32-bit words A, B, C, D, n1, n2, n3, and n4. This subroutine is used in step 4.

$$A \leftarrow n \oplus R[4l], B \leftarrow (n \gg 8) \oplus R[4l+1], C \leftarrow (n \gg 16) \oplus R[4l+2], D \leftarrow (n \gg 24) \oplus R[4l+3]$$

For j from 1 to 2 do the following:

$$P \leftarrow A \wedge 0x000007fc, B \leftarrow B + T[P/4], A \leftarrow (A \gg 9),$$

$$P \leftarrow B \wedge 0x000007fc, C \leftarrow C + T[P/4], B \leftarrow (B \gg 9),$$

$$P \leftarrow C \wedge 0x000007fc, D \leftarrow D + T[P/4], C \leftarrow (C \gg 9),$$

$$P \leftarrow D \wedge 0x000007fc, A \leftarrow A + T[P/4], D \leftarrow (D \gg 9).$$

$$(n1; n2; n3; n4) \leftarrow (D, B, A, C).$$

$$P \leftarrow A \wedge 0x000007fc, B \leftarrow B + T[P/4], A \leftarrow (A \gg 9),$$

$$P \leftarrow B \wedge 0x000007fc, C \leftarrow C + T[P/4], B \leftarrow (B \gg 9).$$

$P \leftarrow C \oplus 0x000007fc, D \leftarrow D + T[P/4], C \leftarrow (C \Rightarrow 9).$

$P \leftarrow D \oplus 0x000007fc, A \leftarrow A + T[P/4], D \leftarrow (D \Rightarrow 9).$

3. Initialize y to be the empty string,

4. Repeat the following:

4.1 Execute the procedure Initialize(n, l, A, C, D, n1, n2, n3, n4).

4.2 For i from 1 to 64 do the following:

$P \leftarrow A \oplus 0x000007fc, B \leftarrow B + T[P/4], A \leftarrow (A \Rightarrow 9), B \leftarrow B \oplus A,$

$Q \leftarrow B \oplus 0x000007fc, C \leftarrow C \oplus T[Q/4], B \leftarrow (B \Rightarrow 9), C \leftarrow C + B,$

$P \leftarrow (P + C) \oplus 0x000007fc, D \leftarrow D + T[P/4], C \leftarrow (C \Rightarrow 9), D \leftarrow D \oplus C,$

$Q \leftarrow (Q + D) \oplus 0x000007fc, A \leftarrow A \oplus T[Q/4], D \leftarrow (D \Rightarrow 9), A \leftarrow A + D,$

$P \leftarrow (P + A) \oplus 0x000007fc, B \leftarrow B \oplus T[P/4], A \leftarrow (A \Rightarrow 9),$

$Q \leftarrow (Q + B) \oplus 0x000007fc, C \leftarrow C + T[Q/4], B \leftarrow (B \Rightarrow 9),$

$P \leftarrow (P + C) \oplus 0x000007fc, D \leftarrow D \oplus T[P/4], C \leftarrow (C \Rightarrow 9),$

$Q \leftarrow (Q + D) \oplus 0x000007fc, A \leftarrow A + T[Q/4], D \leftarrow (D \Rightarrow 9),$

$y \leftarrow y \parallel (B + S[4i - 4]) \parallel (C \oplus S[4i - 3]) \parallel (D + S[4i - 2]) \parallel (A \oplus S[4i - 1]).$

If y is $_L$ bits in length then return(y) and stop.

If i is odd, set $(A, C) \leftarrow (A + n1, C + n2)$. Otherwise, $(A, C) \leftarrow (A + n3, C + n4)$.

4.3 set $L, L \leftarrow L + 1$

3.5 Blum Blum and Shub

This simplest and most efficient complexity-theoretic generator is called the Blum Blum, and Shub generator, after its inventors. The theory behind the BBS generator has to do with quadratic residue modulo n. First find the two large prime numbers p, q, which are congruent to 3 modulo 4. The product of those numbers, n, is a Blum integer. Choose another random integer x which is relatively prime to n compute[5]

$$X_0 = X^2 \pmod n$$

That's the seed for generator.

Now you can start computing bits. The i th pseudo-random bit is the least significant bit x_i where

$$x_i = x_{i-1}^2 \pmod n$$

the most intriguing property of this generator is that you don't have to iterate through all $i-1$ bits to get the i th bit. If you know p and q , you can compute the i th bit directly

b_i is the least significant bit of x_i where

$$x_i = x^{(2^{i-1} \dots 1) \pmod{(p-1)(q-1)}}$$

this property means you can use this cryptographically strong pseudo-random-bit generator as a stream cryptosystem for a random-access file. BBS generator is unpredictable to the left and unpredictable to the right. This meant that given a sequence generated by the generator a crypto analyst couldn't predict the next bit in the sequence or the previous bit in the sequence. The Blum-Blum-Shub pseudorandom bit generator (also known as the $X_0 = X^2 \pmod n$ generator or the BBS generator) is a CSPRNG under the assumption that integer factorization is intractable [1]. It forms the basis for the Blum-Goldwasser probabilistic public-key encryption scheme.

Algorithm Blum-Blum-Shub pseudorandom bit generator

SUMMARY: a pseudorandom bit sequence z_1, z_2, \dots, z_l of length l is generated.

1. Setup. Generate two large secret random (and distinct) primes p and q (cf. Note 8.8), each congruent to 3 modulo 4, and compute $n = pq$.
2. Select a random integer s (the seed) in the interval $[1, n-1]$ such that $\gcd(s, n) = 1$, and compute $x_0 \leftarrow s^2$.
3. For i from 1 to l do the following:
 - 1.1 $x_i \leftarrow x_{i-1}^2$.
 - 1.2 z_i the least significant bit of x_i .
2. The output sequence is z_1, z_2, \dots, z_l .

PREDICTION ALGORITHMS

The following algorithms for next bit prediction will be studied and implemented.

1. A Universal Prediction Algorithm given by Jacob Ziv[2] .
2. Sampled pattern matching (SPM). This algorithm is a modification of Ehrenfeucht-Mycielski pseudo random generator algorithm [3]
3. General next bit prediction algorithm. This algorithm is a NESSIE (New European Schemes for Signature Integrity and Encryption) submission, published recently; [4].

4.1 A Universal Predictor Based on Pattern Matching

We consider a universal predictor based on pattern matching. Given a sequence X_1, \dots, X_n drawn from a stationary mixing source, it predicts the next symbol X_{n+1} based on selecting a context of X_{n+1} . The predictor, called the Sampled Pattern Matching (SPM), is a modification of the Ehrenfeucht-Mycielski pseudo random generator algorithm. It predicts the value of the most frequent symbol appearing at the so-called sampled positions. These positions follow the occurrences of a fraction of the longest suffix of the original sequence that has another copy inside $X_1 X_2 \dots X_n$. In other words, in SPM the context selection consists of taking certain fraction of the longest match. The study of the longest match for loss less data compression was initiated by Wyner and Ziv in their 1989 seminal paper.

The predictor can either be deterministic or random. For deterministic predictors there is a function f_n such that

$$\hat{X}_{n+1} = f_{n+1}(X_1, \dots, X_n)$$

For random predictors, one defines a conditional probability distribution, say

$$q(X_{n+1}, \dots, X_n),$$

and sets

$$\Pr\{ \hat{X}_{n+1} = \hat{x}_{n+1} | X_1 = x_1, \dots, X_n = x_n \} = q(\hat{x}_{n+1} | x_1, \dots, x_n)$$

where X_1, \dots, X_n denote random variables. Finally, we can analyze prediction either in the probabilistic setting or the deterministic setting. In the probabilistic setting the sequence X_1, X_2, \dots is generated by a random source with the underlying probability measure P (usually unknown to us) while in the deterministic setting we consider individual sequences.

4.1.1 Sampled Pattern Matching Predictor

It is assumed that a sequence $x^n := x_1 \dots x_n$ is given. Each symbol x_i belongs to a finite alphabet A of size $V := |A|$. For a fixed integer $K \geq 1$, the algorithm will predict the next K symbols that is, $(\hat{x}_{n+1}, \dots, \hat{x}_{n+K})$. However, paper we carry out the analysis of the algorithm only for $K = 1$.

Let us fix $0 < \alpha < 1$. The SPM prediction algorithm works as follows:

1. Find the largest suffix of x^n whose copy appears somewhere in the string x^n . We call this suffix the maximal suffix and denote its length by D_n . More precisely, $D_n := l$ where l is the largest integer such that $(x_{n-l+1}, \dots, x_n) = (x_{n-i-l+1}, \dots, x_{n-i})$ for some $1 \leq i \leq n$.
2. Take the largest fraction of the maximal suffix of length $k_n := \lceil \alpha D_n \rceil$, that is, the suffix x_{n-k_n+1}, \dots, x_n . Such a fractional suffix occurs more than twice in the original string. Let $L_n \geq 2$ be the actual number of times x_{n-k_n+1}, \dots, x_n appears in the string x^n . Each such an occurrence defines a marker (sub string), and the K positions after a markers are called the marked positions. Finally, by a sampled sequence we mean the sequence composed of all symbols from the K -tuple marked positions. We shall use these notations throughout the paper.
3. Let now $N(x_1, \dots, x_K)$ be the number of non-overlapping K -tuple (x_1, \dots, x_K) occurrences in the sampled sequence. The SPM predictor assigns

$$(\hat{x}_{n+1}, \dots, \hat{x}_{n+k}) = \arg \max N(x_1, \dots, x_k)$$

with a tie broken in an arbitrary manner (e.g., by a random selection). In words, $(\hat{x}_{n+1}, \dots, \hat{x}_{n+k})$ is assigned to the most frequent K-tuple occurring in the sampled sequence.

4.2 An efficient universal prediction algorithm

Let $k_0(X^0_{-N})$ be the largest integer $i \leq N-1$ such that $X^0_{-N} = X_{-i-j}^j$ for some $1 \leq j \leq N-i$

$k_0 = -1$ if X^0 does not appear in the sequence.

Consider the suffix of X^0 : X^0_{-N} .

Let $N = \frac{N}{(1+T_1^2+T_2^2)}$ where T_1 and T_2 are some positive numbers ($T_1 = T_2^2$) and let j be a positive integer $1 \leq j \leq T_2$.

1. Evaluate $K_0(X^0_{-N})$.
2. For each j , denote by i^j the first instant in $X^0_{-(j+1)T_1^2 - i(N-1)}$ for which $X^0_{i^j - K_0(X^0_{-N})} = X^0_{-K_0(X^0_{-N})}$. If no such instant exists, set $i^j = -N - 1$.
3. Predict X_i to be the letter $X \in \mathbf{A}$ that minimizes:

$$\sum_{j=1}^{T_2} \sum_{i=i^j - N - 1}^{i^j} \delta(X_{i+1}, X)$$

4.3 Introduction to GNPB

The next bit predictor theoretical model as presented in [4] can be seen, in short, as an algorithm that, given all the previous bits generated by a particular pseudorandom number generator, can efficiently predict the next bit with higher than chance probability. That is, if we have one particular pseudorandom sequence of bits

PS : $b_0, b_1, b_2, b_3, \dots, b_i$

generated by an algorithm that depends on a number of parameters (i.e. IV's, polynomials, seeds, etc...), a next bit predictor for that particular generator will be one

algorithm that can compute the next bit b_{i+1} given the previous ones with probability greater than $1/2$ without knowing the particular set of parameters used by the generator.

Obviously, some assumptions on the behavior of the pseudorandom number generator are made, and these assumptions make the next bit predictor generated worthless at predicting any different model. These particular assumptions can be called, in Artificial Intelligence (AI) terms, domain knowledge. They make the task of predicting a particular pseudorandom number generator much simpler, but at the cost of making it worthless for predicting other generators. They lack generality.

Our goal is to look for a general method able of finding regularities and patterns in the output of generators and which is general enough to be used in all of them. This is why we called our proposal a general next bit predictor (GNBP), because it could be used for discovering any predictable pseudorandom number generator. Thus it does not assume the generator behavior, so no previous domain knowledge is required.

We pose the next bit predictor in terms of a classification problem as follows. The input is a sequence of n previously perceived bits, and the desired output (the class to be predicted) is the next bit. Therefore, we define it formally as:

Given:

- set of attributes: n attributes corresponding to the values of the each n previous bit.
- set of classes: two classes (0 or 1) that denote the next predicted bit
- set of training instances: set of sequences of $n+1$ bits, in which the first n bits correspond to the values of the previous bits (attribute values), and the last bit is the class. In case one receives as input only a sequence of m bits where $m \gg n$, then one can split the input in $m/(n+1)$ sequences of bits (instances).

Obtain:

a description that can predict the next bit in the sequence for any given sequence of n bits. This is the same as classifying into the 0 class or the 1 class. Let us see with an example how our approach transforms the classical problem description into a classification one:

Theoretical Model

$b_0, b_1, b_2, \dots, b_i \rightarrow$ Try to predict b_{i+1} (1)

Classification Model

Transform the same data contained in (1) into i -framelength attribute/value instances c_i of the form

$c_0 : b_0, \dots, b_{\text{framelength}-1} \rightarrow b_{\text{framelength}}$

$c_1 : b_1, \dots, b_{\text{framelength}} \rightarrow b_{\text{framelength}+1}$

.....

.....

$c_{i \text{ framelength}} : b_{i \text{ framelength}}, \dots, b_{i-1} \rightarrow b_i$

and try to learn from it, and a pattern, search for regularities, etc... using any of the known classification techniques developed by AI researchers. Then, apply this learned description in the prediction of the next bit b_{i+1} , or, analogously, solve the classification problem corresponding to the following instance:

$b_{i \text{ framelength}+1}, \dots, b_i \rightarrow 0 \text{ or } 1 ?$

Obviously, we will make a classification for this data based in the previously seen classifications and the knowledge extracted from them by our algorithm. That is, we will classify the data $b_{i \text{ framelength}+1}, \dots, b_i$ as belonging to class 0 or class 1, depending on which bit we think has more probabilities to follow.

In fact, the theoretical model can be approximated by a classification problem. The main change required is that we are limited by not considering all the past bits at once, but only a given prefixed number of them. This number of past bits we will consider, that can also be seen as the short-term memory of our system, can be called *framelength* and will be very significant for the problem.

The other requisite of the next bit predictor model is that the algorithm must be efficient. The construction of our predictor is quite fast because, once decided the more adequate *framelength* value, it is linear in the size of the processed data. And, once it has

processed all the data set and constructed the decision tree, the classification (prediction) cost is constant.

Some problems that arise when considering this framelenght are worthy to mention. One of these is noise. We will say we have noise when our model, being exactly in the same state, is supposed to predict different values. This, of course, does not happen in the theoretical model because it will never be in the same state, thus the sequence of bits under consideration is always increasing. However, this is not the case in our model because it is perfectly possible that two identical bit sequences of length framelenght will be followed by different bits. In this case it is clear that the prediction needs to remember more bits than the number specified by the framelenght value. That is when we say we have noise. Fortunately, noise is not a difficult problem to minimize because simply by increasing the size of our framelenght variable, we can approximately reduce the amount of this noise exponentially and, for all practical purposes, we can even eliminate it completely.

The other problem of using this framelenght approximation is that we are losing some information. In the theoretical model we always have more or equal information that in our proposed model. We will be always capable of checking the relationship between all the past bits and the next one, while in our model this will be more difficult, but not impossible, because of the framelenght limitation. Only the last framelenght bits will be scanned for a direct correlation with the next one. Anyway, this is a price we have to pay for applying some very powerful techniques used in machine learning. Again, this information loss could be minimized by increasing the framelenght value. The authors strongly believe that this will simply make the performance of our model worse, but not very significantly when using very large framelenght values.

General next Bit Prediction algorithm

This approach is based on Machine Learning technique. The proposed working scheme of this model is :

- 1) 2 numberofinstancestolearnfrom + framelenght-1 consecutive bits the generator to analyze.

- 2) Convert this binary data into 2 numberofinstancestolearnfrom classification examples
- 3) Divide this 2 numberofinstancestolearnfrom examples into halves and name this files numberofinstancestolearnfrom.data and numberofinstancestolearnfrom.test
- 4) Create a file with the format data and call it numberofinstancestolearnfrom.names
- 5) Put C4.5 to work over this three files and tabulate the results.
- 6) It is proposed to use an inductive technique set the value of numberofinstancestolearnfrom
- 7) set the value of framelength.

IMPLEMENTATION DETAILS

5.1 Next bit /symbol prediction algorithm

5.1.1 Sampled Pattern Matching Predictor

This algorithm is good for predicting the next symbol/bit of PRNG if the length of sequence is more than the period of PRNG. It can predict a PRNG which produces a sequence of integers as well as binary.

This algorithm is implemented in such a way that it considers the input sequence of integers only that's why the maximum number it can consider is 32767. If we have a sequence of pseudo random numbers which are greater than 32767 I took a mode of that number with 32767 and apply SPM to predict the next random number. In case of SEAL I have taken a mod of the sequence of unsigned integer (output sequence is in unsigned integer) with some particular number and then I have applied SPM to work with it. If we want to predict the sequence of random bits(0,1) of PRNG which output decimal number then we can convert the decimal output to the sequence of bits as done in the cases of LCG, RC4, SEAL

5.1.2 An efficient universal prediction algorithm

This algorithm is good for predicting the next number of PRNG if the length of sequence is less than the period of PRNG.

This algorithm is implemented in such a way that it considers the input sequence of integers only that's why the maximum number it can consider is 32767. If we have a sequence of pseudo random numbers which are greater than 32767 I took a mode of that number with 32767 and apply Jiv's algorithm to predict the next random number. In case of SEAL I have taken a mod of the sequence of unsigned integer (output sequence is in unsigned integer) with some particular number and then I have applied Jiv's algorithm to work with it. If we want to predict the sequence of random bits(0,1) of PRNG which output decimal number then we can convert the decimal output to the sequence of bits as done in the cases of LCG, RC4, SEAL

5.1.3 General next bit prediction algorithm

This algorithm considers the Input as a binary sequence only and evaluates the PRNG being considered.

This algorithm is implemented by using machine learning technique called decision tree with one standard algorithm called C4.5[9]. If the output sequence of PRNG is of integer number then we will convert that to sequence of bits , 16 bits in case of LCG and SEAL , 8 bits in case of RC4.

5.2 Pseudo Random Number Generators

5.2.1 LFSR

Various variations of LFSR are there varying in length of shift register and the number by which we shift the Register to produce the pseudo random sequence like[1]

- 4-bit Shift Register
- 16-bit Shift Register
- 32-bit Shift Register etc.

I implemented 16 bit and 32 bit shift register with the following shift of bits .

$$X^{15} + X + 1$$

$$X^{32} + X^7 + X^5 + X^3 + X^2 + X + 1$$

5.2.2 RC4

Rc4 generates byte sequence so the generated byte in decimal system is converted into binary system with 8 bits per pseudo random number[5].

5.2.3 SEAL

Seal and LCG is implemented in such a way so that it generated the integer sequence .For generation of binary sequence I converted it into sequence of binary with 16 bits per integer.

RESULTS AND DISCUSSION

Here we consider the optimal prediction error for unknown finite-alphabet Markov sources, for prediction algorithms that make inference about the most probable incoming letter, where the distribution of the unknown source is apparent only via a short training sequence of N letters (i.e. N is a polynomial function of K , the order of the Markov source, rather than the classical case where N is allowed to be exponential in K). A lower bound on the prediction error is formulated for such universal prediction algorithms as shown below. It is demonstrated that its performance is "optimal" in the sense that it yields a prediction-error which is close to the lower-bound on the universal prediction-error as claimed by Jacob Jiv in General next bit prediction algorithm[] in comparison to SPM.

6.1 Sampled Pattern Matching Predictor :

6.1.1 LCG

For LCG if the output sequence is in integers this algorithm predicts with a probability of 1 if the length of the sequence is more than the period(m) of the LCG. This algorithm predicts only if there are cycles in the sequence being considered. But for bad value of seed it predicts the next symbol with a probability higher than 0.5 even if the length of the sequence being considered is less than the period of LCG. If the sequence being considered is binary this algorithm predicts the next bit with a probability higher than 0.5 even if the length of the sequence is less than the period as shown in tables 6.1 and table 6.3 shown below.

6.1.2 LFSR

For LFSR ,SPM predicts the sequence with a probability of 1 if the length of the sequence being considered is higher than the period of LFSR But if we consider the sequence length less than the period of LFSR SPM somehow predicts but with a

probability less than 1 and more than 0.5 . SPM predicts better if we increases the length of the sequences being considered as shown in the table 6.1 and table 6.3.

6.1.3 SEAL and RC4

For SEAL and RC4 it predicts if the length of the sequence is very high as shown in the tables below, as these algorithms are very strong and it seems that there are negligible cycles in the sequence produced by RC4 and SEAL.

But if we consider the sequence in binary system SPM predicts better and as better as we increase the length of the sequence.

6.2 General next bit prediction algorithm

For LCG if the output is integers this algorithm predicts if the length of the sequence is more than the period or nearly equal to the period of the LCG . This algorithm predicts if there are cycles in the sequence being considered. This algorithm predictor predicts better than SPM even if the length of the sequence is less as compared , given to SPM for prediction.

6.2.1 LCG

For LCG if the output sequence is in integers this algorithm predicts with a probability of 1 if the length of the sequence is more than the period(m) of the LCG. This algorithm predicts only if there are cycles in the sequence being considered. But for bad value of seed it predicts the next symbol with a probability higher than 0.5 or close to 1 even if the length of the sequence is less than the period of LCG. If the sequence being considered is binary this algorithm predicts the next bit with a probability higher than 0.5 even if the length of the sequence if less than the period as shown in table 6.1.

6.2.2 LFSR

For LFSR ,SPM predicts the sequence with a probability of 1 if the length of the sequence being considered is higher than the period of LFSR But if we consider the sequence length less than the period of LFSR SPM somehow predicts but with a

probability less than 1. this algorithm predicts better than SPM on the same length sequence being considered as shown in the table 6.1 and table 6.3.

6.2.3 SEAL and RC4

For SEAL and RC4 it predicts if the length of the sequence is very high as shown in the tables below, as these algorithms are very strong and it seems that there are negligible cycles in the sequence produced by RC4 and SEAL.

But if we consider the sequence in binary system it predicts better and as better as we increase the length of the sequence.

PRNG	Parameters	Sequence Length	(Output) Binary / Integers	Total No. of cases	No. of cases of correct Prediction	Prediction Probability
LCG	50 Different Seeds M=9000	10000	Integers	50	50	1
LFSR	50 Different Seeds $X^{15} + X + 1$	10000	Binary	50	26	0.52
LFSR	50 different Seed $X^{32} + X^7 + X^5 + X^3 + X^2 + X + 1$	10000	Binary	50	23	0.46
SEAL	50 different Seed	10000	Binary	50	22	0.44
SEAL	50 different Seed	10000	Integers	50	5	0.1
RC4	50 different Seed	10000	Binary	50	24	0.48
RC4	50 different Seed	10000	Integers	50	6	0.12

Table 6.1: results obtained by predicting PRNG with SPM algorithm

PRNG	Parameters	Sequence length	(Output) Binary/ Integer	No. of cases	No of cases of correct Prediction	Prediction probability
LCG	50 Different Seeds M=9000	10000	Integers	50	50	1
LFSR	50 Different Seeds $X^{15} + X + 1$	10000	Binary	50	28	0.56
LFSR	50 Different Seeds $X^{32} + X^7 + X^5 + X^3 + X^2 + X + 1$	10000	Binary	50	26	0.52
SEAL	50 Different Seeds	10000	Binary	50	4	0.08
SEAL	50 Different Seeds	10000	Integers	50	25	0.5
RC4	50 Different Seeds	10000	Binary	50	27	0.54
RC4	50 Different Seeds	10000	Integers	50	6	0.12

Table 6.2: Results obtained by predicting PRNG with Universal Prediction algorithm.

PRNG	Parameters	Sequence Length	(Output) Binary / Integers	Total No. of cases	No. of cases correct Prediction	Prediction Probability
LCG	50 Different Seeds M=500	510	Integers	50	50	1
LFSR	50 Different Seeds $X^{15} + X + 1$	30000	Binary	50	35	0.7
LFSR	50 Different Seeds $X^{32} + X^7 + X^5 + X^3 + X^2 + X + 1$	30000	Binary	50	31	0.62
SEAL	50 Different Seeds	30000	Binary	50	32	0.64
SEAL	50 Different Seeds	30000	Integers	50	15	0.3
RC4	50 Different Seeds	30000	Binary	50	31	0.62
RC4	50 Different Seeds	30000	Integers	50	20	0.4

Table 6.3: Results obtained by predicting PRNG with SPM algorithm.

PRNG	Parameters	Sequence length	(Output) Binary/Integer	No. of cases	No of cases of correct Prediction	Prediction probability
LCG	50 Different Seeds M=500	510	Integer	50	50	1
LFSR	50 Different Seeds $X^{15} + X + 1$	30000	Binary	50	41	0.82
LFSR	50 Different Seeds $X^{12} + X^7 + X^5 + X^3 + X^2 + X + 1$	30000	Binary	50	33	0.66
SEAL	50 Different Seeds	30000	Binary	50	35	0.7
SEAL	50 Different Seeds	30000	Integer	50	17	0.34
RC4	50 Different Seeds	30000	Binary	50	33	0.66
RC4	50 Different Seeds	30000	Integer	50	21	0.42

Table 6.4: Results obtained by predicting PRNG with Universal Prediction algorithm.

General next Bit Prediction algorithm

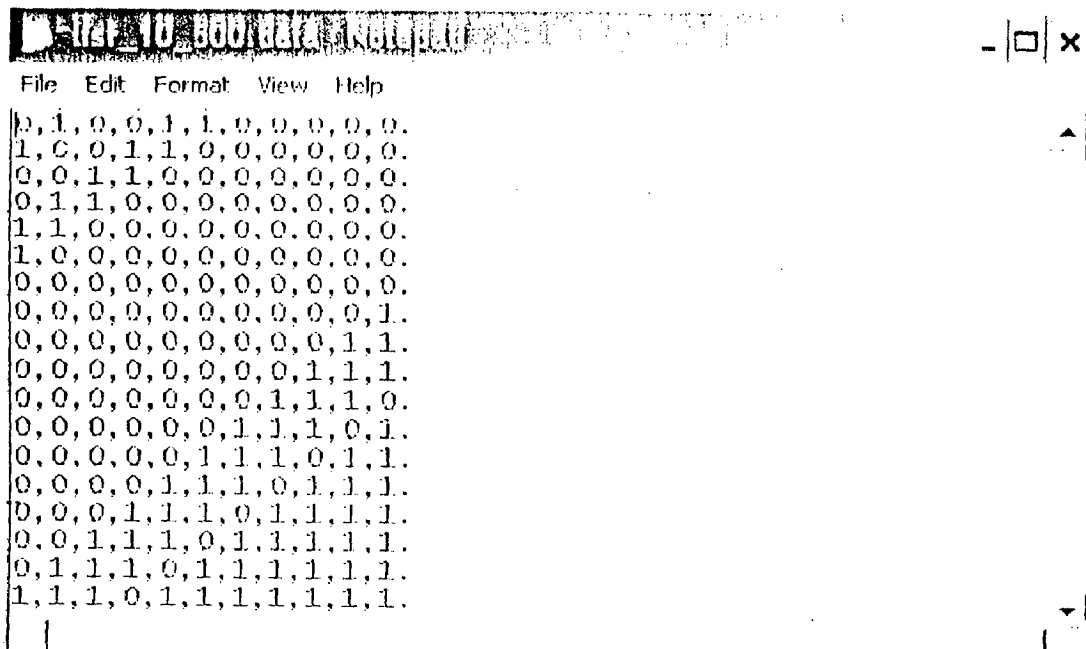
How does C4.5 actually operates?

It needs three files, one called the training set, and another called the test set and a third one in which the common format of the other two is given. This three files have, respectively, the extension *.data, *.test and *.names.

So our next bit predictor will be given two sets of instances: the data and the test sets shown in Figure 6.1 and Figure 6.2 respectively, one names file shown in Figure 6.3. The first one is used to learn the characteristics of a particular generator. The algorithm tries to find a good decision tree for this data. Then it will check the validity of this tree against the other set of instances, the test set, which the algorithm has never previously seen.

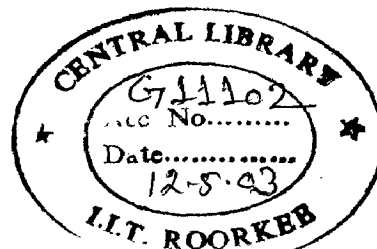
Obviously, we have to provide our next bit predictor with a sufficiently large (and representative) set of data instances to allow it to extract generalizable conclusions that then will be applied and tested against the test set. If we create a data set with non representative instances or without enough data, it won't learn (anyway, no algorithm will do in these conditions) the results are shown in figure 6.5 and 6.6.

Training file will be in the format shown below.



```
File Edit Format View Help
0,1,0,0,1,1,0,0,0,0,0,0.
1,0,0,1,1,0,0,0,0,0,0,0.
0,0,1,1,0,0,0,0,0,0,0,0.
0,1,1,0,0,0,0,0,0,0,0,0.
1,1,0,0,0,0,0,0,0,0,0,0.
1,0,0,0,0,0,0,0,0,0,0,0.
0,0,0,0,0,0,0,0,0,0,0,0.
0,0,0,0,0,0,0,0,0,0,0,1.
0,0,0,0,0,0,0,0,0,0,1,1.
0,0,0,0,0,0,0,0,0,1,1,1.
0,0,0,0,0,0,0,0,1,1,1,0.
0,0,0,0,0,0,1,1,1,0,1.
0,0,0,0,0,1,1,1,0,1,1.
0,0,0,0,1,1,1,0,1,1,1.
0,0,0,1,1,1,0,1,1,1,1.
0,1,1,1,0,1,1,1,1,1,1.
1,1,1,0,1,1,1,1,1,1,1.
```

Figure 6.1 : format of training file



Frame length	10	20	30	40	50	70	100
Before pruning							
Size	1201	8069	4092	7000	8559	7225	6832
%accuracy training	65.5%	68.3%	75.7%	83.6%	89.9%	91.4%	97.7%
%accuracy test	70.1%	85.5%	99.9%	100%	100%	100%	100%
After pruning							
Size	1032	7003	3569	6701	7548	7190	6583
%accuracy training	64.3%	68.4%	75.6%	83.6%	89.9%	91.3%	97.7%
%accuracy test	70.1%	85.5%	99.9%	83.6%	89.9%	91.4%	97.7%

Table 6.5 : results obtained by predicting LCG with GNB algorithm

A predictor for a Linear Feedback Shift Register

In the case of Linear Feedback Shift Register (LFSR) given by $X^{15} + X + 1$ the primitive polynomial $x^{15} + x + 1$. The period of our LFSR generator is $2^{15} - 1$ so we will give it much less than its period as data because we will create a data file (for learning) of only 5 examples and a test file (for testing what has learned) of another different 5 examples.

Results are written in the table below

Frame length	10	20	30
Before pruning			
Size	4	117	1332
%accuracy training	95.5%	97.5%	95.9%
%accuracy test	95.3%	97.0%	95.2%
After pruning			
Size	3	4	7
%accuracy training	95.5%	97.0%	100%
%accuracy test	95.5%	96.9%	100%

Table 6.6 : results obtained by predicting an LFSR with the GNB algorithm

The first thing worth to mention is that I have achieved our desired 100% accuracy much more quickly (with a lower frame length value) than in the LCG case.

Also, we have got this perfect accuracy in a quite surprising manner. In the LCG case there was a much slower convergence to 100% accuracy and here we jump from a relatively far value (95.5%) directly to it. This is because this particular type of generators are much more easily predictable than LCGs and are characterized by simpler rules that are easier to discover. That is exactly what has happened with the predictor. The rule it has discovered after pruning with a framelenght value of 20 can be expressed as:

```

If the bit in the position 6 is 0
    then If the bit in the position 7 is 0
        then classify as 0
        else classify as 1
    else If the bit in the position 7 is 0
        then classify as 1
        else classify as 0

```

That is the same as

class=bit6 XOR bit7

or, analogously,

bit 21=bit 6 XOR bit 7

Adding bit 21 to both sides of the relation we get

bit 21 XOR bit 21 = 0 = bit 6 XOR bit 7 XOR bit 21

So

bit 6 XOR bit 7 XOR bit 21 =0

or

$$x^6 + x^7 + x^{21} = 0$$

and dividing over x^6 we obtain

$$1 + x + x^{15} = 0$$

So general next bit predictor has completely determined the generator and, furthermore, its primitive polynomial by discovering a multiple of the basic relationship

that holds between all the bits produced by this LFSR generator. This is an interesting and rare achievement as it not only predicts the generator, but also gives us definitive clues over its type despite of starting with zero knowledge about it.

The minimum length of sequence required for the next bit predictor to work (predict next bit correctly with a probability better than random guessing) in case of LCG is m . In case of LFSR it depends upon the length of shift register used that is in case of 16 bit shift register it comes out to be 40 bits if we predict using general next bit prediction. But if want to fully predict the LFSR we must have to take the length of the sequence greater than $2^{15} - 1$. for partial prediction with a probability of 0.6 or more we have to take the length greater than 10000 bits.

The typical values (for minimal length) the generators being considered? The typical value LCG take is greater than m and for 16 bit register in LFSR it takes 10000 bits.

RC4 and seal are very strong algorithm and it seems that to predict these algorithm better than tossing a coin we must have to take the length of the sequence to be very high. But LCG and LFSR seems to be weak algorithm and can easily be crypto-analyzed.

CONCLUSION

The optimal prediction error for unknown finite-alphabet Markov sources, for prediction algorithms that make inference about the most probable incoming letter, where the distribution of the unknown source is apparent only via a short training sequence of N letters (i.e. N is a polynomial function of K , the order of the Markov source, rather than the classical case where N is allowed to be exponential in K). A lower bound on the prediction error is formulated for such universal prediction algorithms.

It is demonstrated that the performance of “A Universal Prediction Algorithm” is optimal in the sense that it yields a prediction-error which is close to the lower-bound on the universal prediction-error when compared with SPM. SPM and Universal Prediction Algorithm predicts only in the cases where certain amount of cycles are there in the output of PRNG that's why they predicted LCG and LFSR as they are considered to be weak. But for strong algorithm like SEAL, RC4 they need very large amount of sequence length for prediction. If we have less amount of sequence length we would like to prefer for Universal Prediction Algorithm rather than SPM. We can apply these algorithm to PRNGs to see the testing resistance of existing(old) and new PRNGs to have a good security system. Through these algorithms we can study the patterns of the cycles in the PRNGs sequence and crypto analyze the generators .

It has been proposed that implementation of the next bit predictor theoretical model that uses artificial intelligence techniques to extract knowledge for a given generator and then uses this knowledge to predict its behavior[4]. We have found that the same GNBP model can be useful in predicting completely different types of pseudorandom number generators, namely a LCG and a linear feedback shift register.

This lack of domain knowledge information in our GNBP model is obviously reflected in the form of an increase in the number of bits needed to predict a generator with a given accuracy. Alternatively, the main advantage of this model is that it does not

need to know what kind of generator has been used, so it is completely independent of the particular generator used.

There are many other ML techniques other than C4.5 that might probably lead to interesting results, like the C5.0 algorithm or even some other machine learning paradigms like genetic algorithms. That could be explored in future works for better results.

GNBP approach is very good for testing purposes. It is also an interesting tool to test any behavior that is supposed to be unpredictable, like for example the output of new hash functions, stream ciphers, block ciphers or any other cryptographic primitives. Although the general predictor has been shown predicting with 100% accuracy some different models of generators, we believe that this kind of general predictors will show their best trying to distinguish generators from unpredictable sources. It has no point trying to predict with 100% accuracy generators which are well known and for which we have optimal (or at least, better) predictors. The main advantage of this general predictor could be shown at working with new generators for which we cannot add any domain knowledge, implementing a filter that can distinguish between unpredictable generators and predictable ones for quickly rejecting the latter.

This is by no means, even with the aid of this new tool, a simple task. In fact it has been mathematically proved to be impossible for certain generators. Anyway, I believe that for the vast majority of them, this technique will be very useful.

Another practical use for this model could be to measure predictability, to classify in a systematic way pseudorandom number generators by means of their anti-prediction strength.

REFERENCES

1. Menezes, P. van Oorschot, and S. Vanstone, " Handbook of Applied Cryptography", CRC Press, 1996.
2. Jacob Ziv ,“AN efficient Universal Prediction Algorithm For Unknown Sources With Limited Training Data”, IEEE transactions of information theory, vol. 48, pp 1690-1693,June 2002.
3. Philippe Jacquet,Szpankowski "A Universal Predictor Based on Pattern Matching", IEEE transactions of information theory , vol..48,6 pp.1462-1472, june 2002.
4. Hernandez, J.C, Sierra, J.M.,Mex-Perera, Isasi.P. " Using the General next bit predictor like an evaluation criteria", NESSIE (New European schemes for signature integrity and encryption) submission, June 2001.
5. B. Schneier,"Applied Cryptography" ,John Wiley and Sons, Inc, second edition , 2001.
6. D.Michie,J.R.Quinlan,"Programs for machine learning"Edinburg University Press,1979.
7. T. Mitchell, "Decision Tree Learning", The McGraw-Hill Companies, Inc., 1997, pp. 52-78
8. P. Winston, "Learning by Building Identification Trees", Addison-Wesley Publishing Company, 1992, pp. 423-442.
9. <http://www.cse.unsw.edu.au/quinlan>.