Effective Prediction and its Computational Complexity

Richard Taylor

Australian Defence Science and Technology Organisation
Fern Hill Park ACT 2600 AUSTRALIA

richard.taylor@dsto.defence.gov.au

Abstract

A model for the problem of predicting the outputs of a process, based only on knowledge of previous outputs, is proposed in terms of a decision problem. The strength of this particular formulation of the decision problem follows from the accuracy of its outputs in predicting the outputs of any particular deterministic process, and this predictability is quantified in terms of the number of bits the process may generate, and its length/time complexity. Both upper and lower bounds on the computational complexity of this decision problem are provided

Keywords: Kolmogorov Complexity, Computational Complexity, Prediction.

1 Introduction

A model for the problem of predicting the outputs of a process, based only on knowledge of previous outputs is proposed in Section 2 in terms of a decision problem. In Theorem 1 of Section 3 the accuracy of this model in terms of the outputs of the decision problem(s) matching the outputs of any particular deterministic process is quantified. This is done in terms of the number of bits the process may generate, and its length/time complexity. In Section 4 lower bounds to the computational complexity of any prediction algorithm that is, in a certain defined sense, "approximately accurate" and "almost always approximately accurate", are given in Theorems 2 and 3 respectively. In Section 5 probabilistic approximations to computing outputs of the prediction decision problem are shown to follow from the outcomes of certain sub-problems which are in the class *NP* for certain *f*. These sub-problems are not known to be *NP-complete* however.

The tools developed in this paper are based on the theory of Kolmogorov Complexity (see the book by Li and Vitanyi 1997 and the special journal issue of Gammerman and Vovk 1999 for background).

We shall use the term bit generator to denote a computer program that outputs an infinite binary string. Let P be such a bit generator. Let P_k be the *kth* output bit of P with $[P_k]$ denoting the sequence of the first k output bits $P_1, P_2, .., P_k$ of P. Define the length $LEN(P)$ of a bit generator P, as the number of bits in its representation.

Definition 1 *Define the complexity of a bit generator P and its first k output bits as*

$$COMP(P,k) \equiv LEN(P) + f(t_k^P)$$

where P generates $P_1, P_2, .., P_k$ in time t_k^P, and where f is an integer-valued (non-strictly) increasing function of t_k^P.

As the analysis within this section can be performed just as easily for any such f we need not specify any particular form here, however we shall discuss a suitable form of f later in the paper.

Definition 2 *Define the complexity of any bit string $[a_k] = a_1, a_2, .., a_k$ as*

$$COMP_S([a_k]) \equiv \min_{A \in \Omega} \{COMP(A,k): a_i = A_i, i = 1,..,k\}$$

where the minimum is taken over all bit generators Ω. Thus $COMP_S([P_k]) \leq COMP(P,k)$ (Note the use of the subscript S to distinguish the complexity of strings from bit generators).

Definition 3 *The set of bit generators $S([a_k])$ is defined by $R \in S([a_k])$ if and only if $COMP(R,k)=COMP_S([a_k])$, and $R_i=a_i$, $i=1,...,k$.*

Thus $R \in S([a_k])$ if the first k outputs of R match $[a_k]$, and no bit generator with less complexity than R has this property. The notion of bit generator or computer program can be formalised in terms of reference to a specified Universal Turing Machine (UTM) U, in which a bit generator P is an input string to U, and the outputs of U correspond to the familiar notion of the outputs of P.

2 The Prediction Decision Problem

Let f be some function. Define the prediction problem $PREDICT_f$ of the bit stream $[a_k]$ as,

$PREDICT_f([a_k])$

Input: $[a_k]$

Output: 0 if $|Q_0| \geq |Q_1|$, 1 if $|Q_0| < |Q_1|$, where

$Q_0([a_k]) = S([a_k]) \cap S([a_k],0)$,

$Q_1([a_k]) = S([a_k]) \cap S([a_k],1)$.

In words $PREDICT_f([a_k])$ uses a majority of the next bit outputs from the least complex bit generators that are consistent with $[a_k]$. In order to facilitate Theorem 1 following, we also require that the set of bit generators under consideration do not increase their complexity in generating a *k+1th* bit. Thus Q_0, for example, are those bit generators of $S([a_k])$ that also generate a *k+1th* 0 bit

without addition to their complexity. So if Q_0 is non-empty, then $COMP_S([a_k]) = COMP_S([a_k],0)$.

2.1 Remarks

$PREDICT_f$ is designed to apply to any bit stream whatever, whether generated by a random or deterministic process. Thus no assumptions about the input stream are designed in to the prediction problem. $PREDICT_f$ has a corresponding computable algorithm since the sets $S([a_k])$ can be systematically determined. This is because a bound D for $COMP_S([a_k])$ can always be determined by noting the complexity of a bit generator that simply prints the list $[a_k]$. Let T be the least integer such that $f(T) \geq D$. T and D are then upper bounds on the time and length, respectively, of any bit generator in $S([a_k])$. The members of $S([a_k])$ can thus be effectively determined by testing all bit generators of length up to D, and waiting up to time T for each one, noting the outputs, and the complexity of each. Thus there is an algorithm for generating the outputs of $PREDICT_f$.

Of interest is the accuracy of $PREDICT_f$ when applied to the outputs of a deterministic process. That is how does $PREDICT_f([P_k])$(output) correlate with P_{k+1}. We note that it is an easy matter to construct a bit generator for which $PREDICT_f$ fails to predict the next output bit correctly. Let $[P_k]$ be the first k outputs of P. Construct P^* from P by ensuring P^*_{k+1} is not equal to the output of $PREDICT_f([P_k])$, but otherwise produces the same outputs as P. For example the meta code for P^* might be

While (number of outputs ≠ k)

Run P

Else

Print ($1 \oplus PREDICT_f([P_k])$(output)).

Thus $PREDICT_f([P^*_k])$(*ouput*) produces the 'wrong' prediction for P^*_{k+1}. For a given bit generator P however there is a sense in which the output of $PREDICT_f([P_k])$ is successful as a predictor of the $k+1th$ output of P, when considered over a range of k . We quantify this in terms of the complexity of P and a bound on k.

3 Accuracy

Let B be the number of output bits that scope the area of interest regarding the output generated by P.

Theorem 1 For a given bit generator P, and complexity function f, the proportion of the problems $PREDICT_f([P_k])$, $k=1,..,B$, which produce predictions for the next bit which are "correct" (that is $PREDICT_f([P_k])$(Output) = P_{k+1}) is at least

$$1-\frac{3}{4B}(COMP(P,B))(COMP(P,B)+1)$$
$$=1-\frac{3}{4B}\left(LEN(P)+f(t_k^P)\right)\left(LEN(P)+f(t_k^P)+1\right)$$

Proof Consider the set X of indices between 1 and B for which the prediction problem produces the "wrong" output bit. That is $PREDICT_f([P_k])$(Output) $\neq P_{k+1}$ for $x \in X$. This means that

$$\left|Q_{P_{x+1}}\right| \leq \left|Q_{\overline{P_{x+1}}}\right|.$$

Now the complexity $COMP(R,i)$ associated with the bit generators $R \in S([P_i])$, $i=1,..,B$ is a (non-strictly) increasing integer sequence from within the range 1 to $COMP(P,B)$. It follows that there must be a subsequence of consecutive indices $i=y,y+1,..,z$ for which $S([P_i])$ contains bit generators R in which $COMP(R,i)$ all have the same value j, and which contain at least

$$\frac{2j|X|}{COMP(P,B)(COMP(P,B)+1)}$$

of the elements of X. If this were not the case then we may bound the number of elements of X as less than

$$\sum_{j=1}^{COMP(P,B)} \frac{2j|X|}{COMP(P,B)(COMP(P,B)+1)} = |X|$$

which is a contradiction (this argument is a version of the pigeon-hole principle).

Now $S([P_y]) \supseteq S([P_{y+1}]) \supseteq \ldots \supseteq S([P_z])$, since the generators in these sets all have the same associated complexity.

Also for $x \in X$, $y \leq x \leq z-1$,

$$Q_{P_{x+1}} = S([P_x]) \cap S([P_{x+1}]) = S([P_{x+1}]).$$

Now $Q_{P_{x+1}} = S([P_{x+1}])$, and $Q_{\overline{P_{x+1}}}$ are disjoint subsets of $S([P_x])$, so that $Q_{P_{x+1}} = S([P_{x+1}]) \leq Q_{\overline{P_{x+1}}}$ ensures that the ratio $|S([P_{x+1}])| / |S([P_x])|$ is less than or equal to ½. Thus the ratios,

$$\frac{|S([P_{y+1}])|}{|S([P_y])|}, \frac{|S([P_{y+2}])|}{|S([P_{y+1}])|}, \ldots, \frac{|S([P_z])|}{|S([P_{z-1}])|}$$

are each at most 1, and at least

$$\frac{2j|X|}{COMP(P,B)(COMP(P,B)+1)}$$

of these ratios are less than or equal to ½. By multiplication,

$$\frac{|S([P_z])|}{|S([P_y])|} \leq \frac{1}{2^{2j|X|/COMP(P,B)(COMP(P,B)+1)}}.$$

Since $|S([P_z])| \geq 1$ we have,

$$\frac{|S(P_y)|}{2^{2j|X|/COMP(P,B)(COMP(P,B)+1)}} \geq 1.$$

Since the elements of $S([P_y])$ have complexity j they must each have a length at most j. It follows that $|S([P_y])| \leq 2^{j+1}$. Thus

$$2^{j+1-2j|X|/COMP(P,B)(COMP(P,B)+1)} \geq 1.$$

Rearranging for $|X|$ and noting that $j \geq 1$,

$$|X| \leq \frac{3}{4} COMP(P,B)(COMP(P,B)+1).$$

The result then follows.

3.1 Remarks

In words, if the number of output bits B is sufficiently large in comparison with the complexity $COMP(P,B)$ (broadly $f(t_B^P)$ small in comparison to $\sqrt{B}$) solutions to the prediction problem $PREDICT_f$ have good predictive value. On the other hand it is clear that if the $COMP(P,B)$ approaches the number of output bits B, then at some point no version of the prediction problem could give a good predictive value. In the context of cryptography, $PREDICT_f$ can be thought of as the cryptanalysis problem for stream ciphers, which is universal in the sense that no assumptions about cipher design are built in to the formulation of $PREDICT_f$. From this perspective the predictive failure indicated by Theorem 1 for $COMP(P,B)$ close to B is familiar as the statement of unbreakability of a one-time pad (see the general cryptography reference of Schneier 1996).

3.2 A Trade-off

Notice that there is a trade-off involved in the time component of the complexity function used. Choosing a function f with a smaller time component (f grows more slowly with t) results in a reduced value for $COMP(P,B)$ and hence a better predictive value according to Theorem 1. On the other hand the computational complexity of $PREDICT_f$ is likely to increase since the wait to test membership $S([P_k])$ takes up more time (see the remarks following definition of $PREDICT_f$). In fact we can quantify a relationship between f and the computational complexity of $PREDICT_f$, and this is pursued in Section 4.

3.3 Choice of *f*

At this point it is worth reflecting on the range of functions f that are useful in terms of Theorem 1. Firstly from our definition of $COMP(P,k)$ there is a relationship between bit generators P and their corresponding time behaviour t_B^P, and the time function f that allows Theorem 1 to be usefully applied. Thus for a given function f, for sufficiently large B, the equation

$$1 - \frac{3}{4B}(LEN(P) + f(t_B^P))(LEN(P) + f(t_B^P) + 1) > \frac{1}{2}$$

defines a collection of bit generators, or similarly for a collection of bit generators the equation defines a range of functions f.

In absolute terms t_B^P would seem to be at least of order k since any bit generator that generates k output bits must use at least k time steps to achieve this. On the other hand we know that many interesting functions can use as few as a constant number of operations per output. It is easy for example to construct an infinite family of stream ciphers from linear feedback shift registers of increasing lengths, but all with the same number of feedback taps and the same combining logic (see the book by Rueppel (1986)). Thus t_B^P as low as ck would seem to be relevant where c is a constant. In this case

$$1 - \frac{3}{4B}(LEN(P) + f(t_B^P))(LEN(P) + f(t_B^P) + 1)$$
$$= 1 - \frac{3}{4B}(LEN(P) + f(cB))(LEN(P) + f(cB) + 1) > \frac{1}{2}$$

for sufficiently large B provided $f(x)$ is at most x^α, $\alpha < 1/2$. This upper bound determines general limits on functions f that we shall consider in this paper.

3.4 Simplicity and Variety

It is interesting to identify two important principles, which are integrated in Theorem 1. Favouring the simplest (cf least complex used here) explanation of a set of data in understanding phenomena, and factoring in the variety of views that may be consistent with observations. The former is a well known scientific principal sometimes referred to as Occam's razor, while the latter has been observed as a strength of utilising diverse views rather than narrow thinking (see the book of Klien et al 1995) for a discussion in relation to decision making, for example). Thus $PREDICT_f$ combines simplicity (through minimum complexity), with variety (through the majority function applied to all equally simple bit generators). Furthermore it appears that both of these principles are necessary to proving Theorem 1.

3.5 Related Results

Related prediction functions include those where some single bit generator consistent with $[P_k]$ is used as a basis of prediction. For example Gold (1967) suggests using the least complex bit generator with the minimum lexicographic order. However, in this case it does not seem possible to obtain a useful equivalent to Theorem 1.

Also the halving algorithm (see the article of Littlestone 1988) is a prediction method based on a majority analysis of functions (cf bit generators) consistent with the input data. This requires a finite class of functions from which all functions are drawn. However the relevant class of functions in $PREDICT_f$ (the $S([P_y])$) vary with the length of the initial bits and so the error rate analysis is more complex.

4 Computational Complexity of Prediction

Theorem 1 shows that $PREDICT_f$ is an accurate predictor of generators that are suitably limited in their time growth. We provide lower bounds for the computational complexity of all prediction algorithms that are effective at predicting collections of generators in an approximate sense.

Let S be the set of all finite bit strings.

Definition 4 *Let P be a collection of bit generators. An algorithm* $A: S \rightarrow \{0,1\}$ *is an effective approximate*

predictor over P if for every bit generator P∈P then for some ε>0 and all sufficiently large integers L

$$\frac{\left|\{l \le L : A([P_l]) = P_{l+1}\}\right|}{L} > \frac{1}{2} + \varepsilon .$$

We may now state

Theorem 2 Let P be the collection of bit generators defined by $t_B^P \le g(B)$. Any algorithm that is an effective approximate predictor over P has a computational complexity $C(k)$ that satisfies the bound,

$$\sum_1^k C(i) \ge g(k) \quad \text{for arbitrarily large } k.$$

Proof Assume on the contrary that some algorithm A is an effective approximate predictor over P and has a computational complexity $C(k)$ satisfying

$$\sum_1^k C(i) < g(k) \quad \text{for all sufficiently large } k \ (\text{say } k > K).$$

Define the bit generator G by

$$G_{i+1}=A([G_i])\oplus 1.$$

Thus G is designed to disagree with the predictions generated by A.

Also

$$t_k^G \le \sum_1^k C(i) < g(k) \quad \text{for } k > K.$$

Thus $G\in P$. Since A is an effective approximate predictor over P then for some $\varepsilon > 0$ and all sufficiently large integers L

$$\frac{\left|\{l \le L : A([G_l]) = G_{l+1}\}\right|}{L} > \frac{1}{2} + \varepsilon .$$

This however contradicts the fact that G is designed to disagree with the predictions of A.

This contradiction proves the theorem.

4.1 Incomplete Approximations

Note that Theorem 2 uses a self referencing argument to construct a generator G from the prediction algorithm A that A fails to approximately predict. We can extend the counter example G to an infinite class, thereby generalising Theorem 2. We shall need some definitions, and a preliminary lemma.

In what follows we shall consider two bit generators G and H to be *different* if their outputs differ.

Definition 5 *Let P be an infinite class of bit generators, and A the class of all bit generators. We say that P includes almost all bit generators if*

$$\frac{\left|\{P \in \mathbf{P} : LEN(P) \le L\}\right|}{\left|\{P \in \mathbf{A} : LEN(P) \le L\}\right|} \to 1$$

As $L \to \infty$.

We also have

Definition 6 *P has positive density among all bit generators if there exists some* $\varepsilon > 0$ *for which*

$$\frac{\left|\{P \in \mathbf{P} : LEN(P) \le L\}\right|}{\left|\{P \in \mathbf{A} : LEN(P) \le L\}\right|} > \varepsilon$$

for all sufficiently large L.

Lemma 1 Let $A{:}S \to \{0,1\}$ be any algorithm. Let V be the class of finite bit strings of the form $V=V_1, V_2, \ldots, V_k$, k a positive integer, such that

$$\sum_{i=1}^k V_i \ge \frac{k}{2}.$$

Clearly at least half of all bit strings of length k satisfy this inequality. Let G be the class of bit generators, each $G\in G$ defined for each $V=V_1, V_2, \ldots, V_k$ by

$$G_{i+1}=A([G_i])\oplus V_j \text{ where } j\equiv i(\text{mod}k),\ 0\le j\le k-1.$$

Then

1) the class G has positive density among all bit generators,

2) for each integer L and $G\in G$

$$\frac{\left|\{l \le L : A([G_l]) = G_{l+1}\}\right|}{L} \le \frac{1}{2},$$

3) for some constant d_G (dependent on G)

$$compute(G_{i+1}) \le compute(A([G_i])+d_G$$

where $compute(G_{i+1})$ denote the time to compute G_{i+1} from G_i, and $compute(A([G_i])$ the time to compute $A([G_i])$ from G_i.

Proof 1) Consider all strings V_L of V of length L for some fixed L. Note that different elements of V_L produce different elements of G. To see this let P and Q be two elements of V_L, and G and H defined correspondingly by

$$G_{i+1}=A([G_i])\oplus P_i,\ H_{i+1}=A([H_i])\oplus Q_i.$$

Since P and Q are distinct there is some minimum i for which $P_i \ne Q_i$. It follows that

$$G_{i+1}=A([G_i])\oplus P_i \ne H_{i+1}=A([H_i])\oplus Q_i.$$

Thus G and H produce different outputs and so are distinct.

Also by the construction of G from V there is some constant e (dependent on A) for which

$$LEN(G)\le e+LEN(V)$$

for each $V\in V$ and corresponding $G\in G$.

Thus

$$
\begin{aligned}
&\left|\{G \in \mathbf{G} : LEN(G) \le M\}\right| \\
&\ge \left|V_{M-e}\right| \\
&\ge 2^{M-e-1} \\
&= \frac{1}{2^{e+2}} 2^{M+1} \\
&\ge \frac{1}{2^{e+2}} \left|\{P \in \mathbf{A} : LEN(P) \le M\}\right|.
\end{aligned}
$$

This proves 1).

2) This follows by the construction of G from V and the definition of V.

3) This follows from the fact that for each G the corresponding bit string V is finite.

This completes the proof.

Theorem 3 Let P be the collection of bit generators defined by $t_B^P \le g(B)$. Any algorithm that is an effective approximate predictor over almost all of P has a computational complexity $C(k)$ that satisfies the bound,

$$\sum_1^k C(i) \ge g(k) - k\log(k) \quad \text{for arbitrarily large } k.$$

Proof The proof proceeds by contradiction. Assume contrary to the premise of the theorem that some algorithm A is an effective approximate predictor over almost all of P, and that A has a computational complexity $C(k)$ satisfying

$$\sum_1^k C(i) < g(k) - k\log k \quad \text{for all sufficiently large } k \ (\text{say } k > K).$$

Let G be the class of bit generators as defined in Lemma 1 (with the algorithm A from the premise of the proof). Let $G \in G$. Using Lemma 1 – Part 3,

$$
\begin{aligned}
t_B^G &\le \sum_1^B \left[C(i) + d_G\right] \\
&\le d_G B + \sum_1^B C(i) \\
&\le B\log B + \sum_1^B C(i) \quad \text{for } B > e^{d_G} \\
&< g(B).
\end{aligned}
$$

Thus $G \in P$ and so $G \subseteq P$. By Lemma 1 – Parts 1 and 2, G constitutes a subset of P of positive density among all bit generators (Lemma 1 part 1), that A fails to be an effective approximate predictor over. This contradicts our initial assumption about A however, and proves the Theorem.

We note that Theorem 3 still holds if *log(k)* is replaced by any unbounded increasing function of k. We also state the contrapositive form of this Theorem.

Theorem 3 – Contrapositive Form Let P be the collection of bit generators defined by $t_B^P \le g(B)$. Let $A{:}S \to \{0,1\}$ be any algorithm with a computational complexity $C(k)$ that satisfies the bound,

$$\sum_1^k C(i) < g(k) - k\log(k) \quad \text{for arbitrarily large } k.$$

There is then a subset of P of positive density among all bit generators that A fails to be an effective approximate predictor over.

4.2 Upper Bounds

We complement the previous results with upper bounds on the computational complexity of algorithms for $PREDICT_f$.

It shall prove useful to bound both the execution time and the length of any element of $S([a_k])$. Note firstly that the program below is of order k in both length and execution time.

Print a_0

Print a_1

Print a_2

.

.

.

Print a_k.

We thus have,

Proposition 1 For some constant c (depending on the particulars of the chosen Universal Turing Machine U),

$$COMP_S([a_k]) \le ck.$$

It follows that any $R \in S([a_k])$ must have $LEN(R) + f(t_k^R) \le ck$. This in turn means that,

Proposition 2

$$LEN(R) \le ck,$$
$$t_k^R < f^{-1}(ck).$$

Thus $PREDICT_f([a_k])$ can be computed by producing all bit generators of length up to ck, and waiting up to time $f^{-1}(ck)$ for each to produce $[a_k]$ and one more output bit. The following is an immediate consequence of Proposition 2,

Proposition 3 There is an algorithm that computes $PREDICT_f$ that has a computational complexity of order

$$2^{ck} f^{-1}(ck).$$

We may combine Theorems 1 and 2 and Proposition 3 to give

Theorem 4 Let A be an algorithm that is most efficient in computing $PREDICT_f$. Then A has a computational complexity *C(k)* that satisfies

$$\frac{1}{k} f^{-1}\left(\sqrt{\frac{k}{2}}\right) \le C(k) \le 2^{ck} f^{-1}(ck).$$

Proof The upper bound is Proposition 3. By Theorem 1 if

$$f(t_B^P) < \sqrt{\frac{B}{2}}$$

$PREDICT_f$ is an effective approximate predictor of P according to

$$\frac{\left|\{l \le B : PREDICT_f([P_l])(output) = P_{l+1}\}\right|}{B}$$
$$\ge 1 - \frac{3}{4B}(LEN(G) + \sqrt{\frac{B}{2}})(LEN(G) + \sqrt{\frac{B}{2}} + 1)$$
$$> \frac{9}{16} \text{ for large enough } B.$$

Thus Theorem 2 applies with

$$g = f^{-1}(\sqrt{\frac{k}{2}})$$

and so

$$\sum_1^k C(i) \ge f^{-1}(\sqrt{\frac{k}{2}}).$$

Assuming the complexity function is increasing we have the lower bound.

5 Probabilistic Approximations in NP

Following the discussion of Section 3.3 we shall consider the family $f(x)=x^\alpha$ for α <1/2. This minimises the computational complexity of computing $PREDICT_f$ while maintaining its predictive power by Theorem 1. As it stands, notice that the set of bit generators considered in the evaluation of $PREDICT_f([P_k])$, that is Q_0 and Q_1, may be exponential (in k) in size. We show how to probabilistically approximate the outputs of $PREDICT_f$ from a series (polynomial in number) of sub-problems, each of which is in *NP*. Thus if *NP=P* $PREDICT_f$ can be probabilistically approximated by a polynomial time algorithm. We do not know however whether the sub problem is *NP-complete* (see the book by Garey and Johnson 1979 for a guide to NP-completeness).

In this case Theorem 1 shows that $PREDICT_f$ provides "correct" outputs for any particular bit generator P according to the probability lower bound of

$$1 - \frac{3}{4B}(LEN(P) + (t_B^P)^\alpha + 1)(LEN(P) + (t_B^P)^\alpha + 2).$$

This probability bound is close to 1, for large B, provided $t_B^P \le B$ (see the discussion in Section 3.3). In the remainder of this section we shall omit the reference to f and assume it is of the form $f(x)=x^\alpha$ for α <1/2.

5.1 The Sub-problem

Identify each bit generator with an integer corresponding to the binary value of the string representing the corresponding computer program. For *a, b, l,* integers, $\delta \in \{0, 1\}$, define $V_\delta([a_k],r,s,l)$ by

$R \in V_\delta([a_k],r,s,l)$ if and only if

1) R corresponds to an integer between r and s inclusive,

2) $COMP(R,k)=COMP(R,k+1) \le l$,

3) $R_i=a_i$, $i=1,\ldots,k$, and $R_{i+1}=\delta$.

We may then define the following sub-problem of *PREDICT*

***PREDICT_SUB*($[a_k]$, *r, s, l,* δ)**

Input: $[a_k]$, *r, s, l,* δ

Output: *Yes if* $V_\delta([a_k],r,s,l) \ne \phi$,

No if $V_\delta([a_k],r,s,l) = \phi$.

We now describe how answers to this sub-problem can be used to estimate the outcome of PREDICT($[a_k]$).

5.2 Complexity Calculation

We first note that if $COMP_S([a_k]) = COMP_S([a_k], \delta)$ then the value of this integer (say L) will be useful in the estimation of $|Q_\delta|$, and can be determined through a binary search method using answers to the *PREDICT_SUB* problem.

To see this begin with the problem *PREDICT_SUB*($[a_k]$, 0, 2^{Ck}, Ck, δ). If the answer is "No" then $COMP_S([a_k]) \ne COMP_S([a_k], \delta)$, and so Q_δ is empty. If the answer is "Yes" then $L=COMP_S([a_k]) = COMP_S([a_k], \delta) \le Ck$. In the latter case now consider the problem *PREDICT_SUB*($[a_k]$, 0, 2^{Ck}, $Ck/2$, δ). If the answer in "No" then $Ck/2 \le L \le Ck$. If the answer is "Yes" then $0 \le L \le Ck/2$. Thus the interval containing L can be successively halved, so that in $log(Ck)$ steps the value of L is determined.

5.3 Approximations

We demonstrate how to estimate the ratio of the sizes of the sets Q_δ used in the *PREDICT* problem from the judicious use of the answers to the *PREDICT_SUB* problem. Bear in mind that these sets may have anywhere from zero to exponentially many elements. In the following we refer to a constant *LIM* which can be arbitrarily chosen, but determines the accuracy of the approximation process. In Figure 1 we depict a tree structure representing a collection of *PREDICT_SUB* problems. In the first stage the space of relevant bit generators is divided into *LIM* subsets and their corresponding *PREDICT_SUB* problems. Let N of these problems have a YES answer (depicted by clear boxes). If N=0 then Q_δ =0. Assume therefore that N>0. In stage 2 each of the N sub-problems is further reduced in scope by randomly choosing the lower or upper halves of the set space. This is repeated a maximum number of times D while the number of *PREDICT_SUB* problems at depth D with YES answers (say M) is greater than or equal to $N/2$. We then estimate $|Q_\delta|$ by,

$$EST(|Q_\delta|) = \frac{M}{N} 2^D.$$

If $EST(|Q_0|) \geq EST(|Q_1|)$ we estimate the output of *PREDICT* as 0, otherwise we estimate the output as 1.

Notice in the above construction that D is at most Ck. Thus the total number of *PREDICT_SUB* problems generated is at most $LIM \times Ck$.

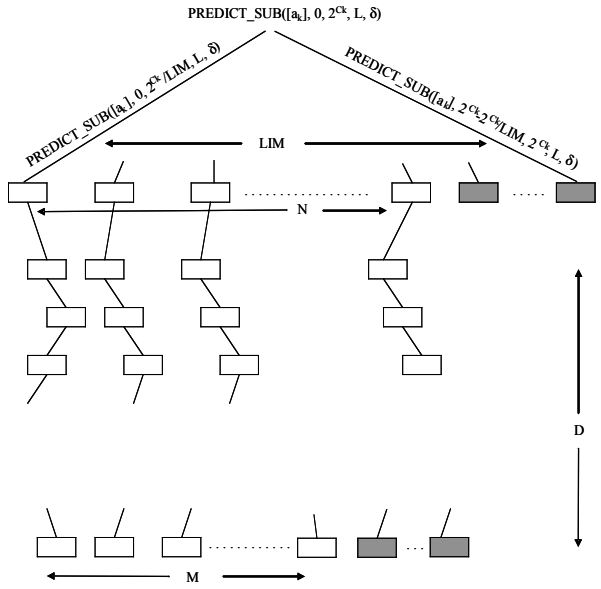

Figure 1: Sub-problem tree

Theorem 4 For $0 \leq r, s \leq 2^{Ck}$, $l \leq Ck$, *PREDICT_SUB*($[a_k]$, r, s, l, δ) is in *NP*.

Proof If the output of *PREDICT_SUB* is "Yes", then there is some element R of V_δ. A certificate consists of running R, and verifying that it satisfies,

1/ R corresponds to an integer between a and b inclusive,

2/ $COMP(R,k) = COMP(R,k+1) \leq l$,

3/ $R_i = a_i$, $i=1,\ldots,k$, and $R_{i+1} = \delta$.

By Proposition 2 this takes at most time $C^{l/\alpha} k^{l/\alpha}$.

5.4 Discussion

Unfortunately, for any fixed $\alpha < 1/2$ *PREDICT_SUB* is not known by the author to be in *P* or to be *NP-complete* (supported by private communications of Allender and also Buhrman 2005). In particular we note that the algorithmic complexity of *PREDICT_SUB*, as a building block for *PREDICT*, bounds the hardness or breakability of any practical (generates outputs in a timely fashion) stream cipher system. Thus practical stream ciphers exist to the extent that *PREDICT_SUB* is computationally infeasible. In this way the computational complexity of *PREDICT_SUB* captures the notion of the existence of a secure stream cipher, or pseudorandom generator, which is complementary to the approach of defining security in terms of properties a generator must possess (see the article of Yao 1982).

6 References

Allender, E. (1992): Applications of Time-Bounded Kolmogorov Complexity in Complexity Theory, *Kolmogorov Complexity and Computational Complexity*, O. Watanabe ed., Springer Verlag.

Allender, E. (2005): private communication.

Buhrman, H. (2005): private communication.

Garey, M and Johnson, D. (1979): *Computers and Intractability: A guide to the Theory of NP-Completeness*, W. H. Freeman and Co.

Gammerman, A and Vovk, V. (1999): eds, Kolmogorov Complexity (Special Issue Editorial), *The Computer Journal*, Vol. 42, No. 4.

Gold, E. M. (1967): Language Identification in the limit. *Inf. Control* 10, pp 447-474.

Klien, G., Orasanu, J., Calderwood, R. and Zsambok, C. (1995): *Decision Making in Action: Models and Methods*, Ablex publishing corporation, Norwood, New Jersey Second Printing .

Levin, L. (1973): Universal sequential search problems, *Problems Inform. Transmission* 9, pp 265-266.

Levin, L. (1984): Randomness conservation inequalities; *Information and independence in mathematical theories, Inform. and Control* 61, pp 15-37.

Littlestone, N. (1988): Learning Quickly When Irrelevant Attributes Abound: A New Linear-Threshold Algorithm, *Machine Learning 2*: pp 285-318.

Li, M. and Vitanyi, P. (1997): *An Introduction to Kolmogorov Complexity and Its Applications* (Second Edition), Springer, New York.

Rueppel, R. (1986): *Analysis and design of stream ciphers*, Springer-Verlag, Berlin.

Schneier, B. (1996): *Applied Cryptography* (Second Edition), John Wiley and Sons.

Yao, A. (1982): Theory and applications of trapdoor functions. Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science, pp 80-91.