Independent Unbiased Coin Flips From a

Correlated Biased Source: a Finite State Markov Chain.

*Manuel Blum**

Department of Electrical Engineering and Computer Sciences; University of California at Berkeley, 94720

Abstract: von Neumann's trick for generating an *absolutely* unbiased coin from a biased one is this:

1. Toss the biased coin twice, getting 00, 01, 10, or 11.
2. If 00 or 11 occur, go back to step 1; else
3. Call 10 a H, 01 a T.

Since p[H] = p[1]*p[0] = p[T], the output is unbiased. Example: *00 10 11 01 01* → H T T.

Peter Elias gives an algorithm to generate an independent unbiased sequence of Hs and Ts that nearly achieves the Entropy of the one-coin source. His algorithm is excellent, but certain difficulties arise in trying to use it (or the original von Neumann scheme) to generate bits in expected linear time from a Markov chain.

In this paper, we return to the original one-coin von Neumann scheme, and show how to extend it to generate an independent unbiased sequence of Hs and Ts from any Markov chain in expected linear time. We give a right and wrong way to do this. Two algorithms A and B use the simple von Neumann trick on every state of the Markov chain. They differ in the time they choose to announce the coin flip. This timing is crucial.

1. Introduction

Von Neumann has given an algorithm (algorithm vN) for getting an *absolutely* unbiased sequence of coin flips from the flips of a biased coin:

Algorithm vN:

Input: A sequence of 0s and 1s, denote it $\sigma_1\sigma_2 \cdots$.

Output: A sequence of Hs and Ts.

For i = 1, 2, ...

do

If $\sigma_{2i} = 1$ and $\sigma_{2i} \neq \sigma_{2i-1}$ **then** output H;

If $\sigma_{2i} = 0$ and $\sigma_{2i} \neq \sigma_{2i-1}$ **then** output T.

od

If the input to algorithm vN is a sequence of 0s and 1s obtained from independent flips of a *biased* coin, then the output is a sequence of independent and *unbiased* Hs and Ts. This is because the probability of producing an H is p*q while that of producing a T is q*p. Here, pr[σ_i = 1] = p for all i, and p + q = 1. Algorithm vN works *without* knowledge of the bias, p. This is surprising, and also useful for applications.

We are interested in producing *independent* unbiased coin flips from a *dependent* biased source, an n-state Markov Chain. The Markov Chain (MC) is a finite set of labelled states $S_1 \cdots S_n$. Out of each state S_i there emerge two arrows, one labelled (1, p_i), the other (0, q_i), where p_i and q_i are probabilities that sum to 1. Each arrow goes to some (arbitrary) state of the MC. The MC produces an infinite sequence $\sigma_1\sigma_2 \cdots$ of 0s and 1s as follows:

> Initially the MC is in the start state, S_1. Whenever it is in state S_i, it selects a 1 with probability p_i (which depends just on the state S_i), else a 0, outputs the selected symbol (1 or 0), then goes to the state indicated by that symbol's arrow.

In this paper, we initially assume that we know the state diagram of the MC in all details *except* that the specific probabilities $\{p_i\}$ attached to the arrows may be omitted. An example where the state diagram of the MC and the start state are known is the (not unreasonable) case where each bit (1 or 0) produced by the source is selected with a probability that depends on just the preced-

*Supported in part by the National Science Foundation under grant MCS 82-04506.

0272-5428/84/0000/0425$01.00 © 1984 IEEE

ing k bits. In that case, the MC has 2^k states, one for every string of k bits (the last k bits produced by the source). Later, we show it suffices to know just an upper bound on the number of states in the MC in order to produce Hs and Ts in expected linear time (i.e., in a constant expected number of steps per output bit). If a bound is unknown, then it is still possible to produce Hs and Ts that are eventually independent and unbiased, but not in expected linear time.

It is easy to see that it is possible in principle to generate independent unbiased bits from a MC: use algorithm vN on the symbols produced by the MC *just* when it exits some particular state, throwing out all symbols produced by the MC when it exits other states. Of course, this is wasteful of the states. Elias [ ] has suggested a way to use all the symbols produced by a MC. His algorithm approaches maximum possible efficiency, but it does not produce bits in expected linear time.

We describe two algorithms, A and B, that make moderately good use of the MC: both have an efficiency* that is 1/4 the entropy of the MC when all probabilities are 1/2. These algorithms do as well as vN does (on a 1-state MC), which is not surprising since both are natural extensions of vN. Algorithms A and B both treat each state of a MC like a von Neumann coin, generating a H (or T) whenever a state of the MC is exited for the $2t^{th}$ time, $t \in \{1, 2, \ldots\}$, provided the $2t\text{-}1^{th}$ and $2t^{th}$ exits are 10 (or 01). Algorithm A outputs the H (or T) as soon as 10 (or 01) is produced by a state. Algorithm B, however, is patient: it waits and outputs the H (or T) later, when the state that produced 10 (or 01) is reentered. The difference (between A and B) is crucial. A is bad while B is good. That is to say, A does not produce independent unbiased output bits while B does. We give faulty "proofs" that A and B are both good. We then exhibit a MC for which algorithm A is bad: it almost *never* produces the quadruple HHTT! Finally, we prove that algorithm B *is* good for every MC.

*Elias defines the *efficiency* to be the expected value of the ratio of output symbols to input symbols. The entropy is defined in the usual way [see Elias]. As expected, the entropy of the source is the maximum possible achievable efficiency.

2. Algorithms

Algorithm A:

Input: An MC (except that probabilities need not be attached to the arrows); A sequence of 0s and 1s produced by that MC.

Output: A sequence of Hs and Ts.

Begin:

Comment: For each state of MC, this algorithm stores one of the 3 symbols λ, 0, 1, called the *state-bit* (though state-trit might be more appropriate). State-bit(S_i) is a memory of the last exit (0 or 1) taken out of state S_i if that was the 1^{st}, 3^{rd}, $\cdots$ such exit; it is λ otherwise. (This is the information needed for this algorithm to treat S_i like a von Neumann coin.)

Comment: The bit produced upon exiting a state is called the *exit-bit*.

Initialize: Set state-bit(S_i) = λ for all i. Set i := 1 [**Comment:** Enter state S_1].

Repeat:

do

1. Set exit-bit := next input bit (i.e., the next bit, 0 or 1, produced by the MC).

2. **Case:**

   i) **If** state-bit(S_i) = λ **then** set state-bit(S_i) := exit-bit.

   ii) **If** state-bit(S_i) = 0 or 1 **then**

   case:

   a) **If** exit-bit = state-bit(S_i) **then** set state-bit(S_i) := λ.

   b) **If** exit-bit $\neq$ state-bit(S_i) **then**

   If <state-bit(S_i), exit-bit> = <1,0> **then** output H; **else** [**Comment**: <state-bit(S_i), exit-bit> = <0,1>] output T.

3. Set i := index of (new) state gotten to by following the exit arrow (arrow labelled by exit-bit) out of (old) state S_i.

od

End

Algorithm B:

Input: An MC (except that probabilities need not be attached to the arrows); A sequence of 0s and 1s produced by that MC.

Output: A sequence of Hs and Ts.

Begin:

Comment: Each state can store any one of the 5 symbols λ, 0, 1, H, T, called the *state-bit* (though state-quint might be more appropriate). State-bit(S_i) is a memory of the last exit (0 or 1) taken out of state S_i if that exit was the 1^{st}, 3^{rd}, $\cdots$ such exit; otherwise it is a memory of the last two exit bits taken out of state S_i: H if they are (in order) 10, T if 01, λ if 00 or 11. (This is the information needed to output the Hs and Ts "patiently.")

Comment: As usual, the bit produced upon exiting a state is called the *exit-bit*.

Initialize: Set state-bit(S_i) = λ for all i. Set i := 1.

Repeat:

do

1. **If** state-bit(S_i) = H (or T), **then** output H (or T) **and** set state-bit(S_i) := λ.

Comment: define the *state-memory* at this point *(after* the above output, if any, has been produced), as the vector of state-bits: <state-bit(S_1),...,state-bit(S_n)>.

2. Set exit-bit := next input bit (i.e., the next bit, 0 or 1, produced by the MC).

3. **Case:**

i) **If** state-bit(S_i) = λ **then** set state-bit(S_i) := exit-bit.

ii) **If** state-bit(S_i) = 0 or 1 **then**

case:

a) **If** exit-bit = state-bit(S_i) **then** set state-bit(S_i) := λ.

b) **If** exit-bit $\neq$ state-bit(S_i) **then**

If <state-bit(S_i), exit-bit> = <1,0> **then** set state-bit(S_i) := H; **else** [**Comment**: <state-bit(S_i), exit-bit> = <0,1>] set state-bit(S_i) := T.

4. Set i := index of (new) state gotten to by following exit arrow (arrow labelled by exit-bit) out of (old) state S_i.

od

End

3. Pseudo-Theorems and Theorems

We now give a faulty "proof" that algorithms A and B are good.

Pseudo-Theorem:

Let MC be a finite state Markov Chain such that with probability 1 every state of MC is visited infinitely often. Algorithms A and B are good in the sense they produce independent unbiased coin-flips from the output of MC.

Pseudo-proof: Each state of the MC may be viewed as a biased coin. When a coin is flipped it produces a 0 or 1 with some bias, then directs the MC to another coin to flip. Algorithm A applied to the MC may be viewed as doing the following:

1. flip coin S_i and get 0 or 1;
2. if this was a 1^{st}, 3^{rd}, ... flip of coin S_i, output nothing.
3. if this was a 2^{nd}, 4^{th}, ... flip of coin S_i, either
   output nothing (with some probability p) or
   output H or T (with equal probabilities (1-p)/2).

Each output of an S_i is equally likely to be H or T. After the coin comes up 0 or 1, the MC is directed to a state where a coin is flipped and the argument reapplied recursively.

Pseudo-qed

Theorem A (Algorithm A is bad):

Let MC be the Markov Chain consisting of **k** states, $S_1, \cdots, S_k$; the 0-arrow out of each state has associated probability $q_i = 1-\epsilon$ and maps the state back to itself; the 1-arrow out of each state has associated probability $p_i = \epsilon$ and maps S_i to S_{i+1} for i < **k**, S_k to S_1.

When Algorithm A is applied to the sequences generated by this MC:

1. (Algorithm A's output sequence is initially biased): for $\epsilon \to 0$ (alternatively: $\epsilon = 1/2$), algorithm A can be expected to output nearly k/2 Hs (nearly k/3 Hs) before outputting its first T.

2. (Algorithm A generates dependent sequences): Let the t^{th} output of A be denoted H_t or T_t. For fixed k $>$ 1, $\mathrm{pr}[T_{t+1}T_{t+2} \mid H_{t-k+1} \cdots H_t] \to 0$ as $\epsilon \to 0$. In particular, for k=2, the probability that MC will output 2 successive Ts given that the last 2 symbols that it output were both Hs is asymptotic to 0 (it should be asymptotic to 1/4).

Proof:

1. Easy to see.

2. We prove this for the special case k = 2.

First note that all Hs and Ts are associated with occurrences of 1-arrow transitions. A T can occur only in case a 0-transition from S_i to S_i is followed by a 1-transition out of S_i. We say that T occurs when the 1-arrow *exits* S_i. Similarly, an H can occur only in case a 1-transition into S_i is followed by a 0-transition from S_i back to S_i. We say that H occurs when the 1-arrow *enters* S_i (though technically it occurs a moment later).

Most (at least ($1-\epsilon$) fraction) of the 1-transitions are followed by 0-transitions. Call these the *likely* 1-transitions. Call an $H_{t-1}H_t$ pair *likely* if a pair of consecutive 1s (11) does not occur between the 1s that produce H_{t-1} and H_t. It is an immediate consequence of the Technical Lemma below and the fact that the fraction of unlikely 1-transitions is asymptotically zero (goes to zero as $\epsilon \to 0$), that the fraction of all t such that the $t-1^{th}$ and t^{th} outputs are "likely" $H_{t-1}H_t$ pairs is asymptotically $\geq 1/4$. In the next paragraph, we focus just on these likely $H_{t-1}H_t$ pairs.

Suppose, without loss of generality, that H_{t-1} is output by A upon entrance of MC to S_1. Since this output cannot be followed by a T, the next 1-transition (from S_1 to S_2) must leave state-bit(S_1) = 1. Since by assumption the 1-transition into S_2 is a likely one, either H_t is generated upon entrance to S_2 (if state-bit(S_2) = 1), or else (if state-bit(S_2) = λ) the next 1-transition out of S_2 leaves state-bit(S_1) = 1. In that case, H_t is generated upon entrance to S_1. In either case, H_t is generated upon entrance to one one of the states at a moment in which the other state has its state-bit set to 1.

Now observe what this means for generating $T_{t+1}T_{t+2}$. After generating H_t, A can generate at most one symbol, T_{t+1}, before entering the other state, at which time it will almost certainly (with probability $\geq 1-\epsilon$) generate H. Therefore, either the symbol generated at time t+ 1 or t+ 2 will be H.

It follows that $\mathrm{pr}[T_{t+1}T_{t+2}] \to 0$.

Qed

Technical Lemma (for Theorem A):

Apply Algorithm A to the MC of the above theorem with k=2 states. Then $\lim_{t\to\infty} \Pr[H_{t-1}H_t] \geq 1/4$.

Proof: $\Pr[H_{t-1}H_t] = \Pr[H_t \mid H_{t-1}]*\Pr[H_{t-1}]$.

1. $\Pr[H_t \mid H_{t-1}] \geq 1/2$ for all t: Without loss of generality, assume that H_{t-1} is generated by S_1, at which time state-bit(S_1) is set to λ. The probability that S_1 generates an H the next time it generates anything at all (H or T) is 1/2 (because S_1 generates independent unbiased sequence of Hs and Ts). Also, after H_{t-1} is generated, the probability that H is the next output generated by S_2 is $\geq 1/2$, since state-bit(S_2) = λ or 1 at the time H_t is output. Therefore, the t^{th} symbol output by B is H_t with probability $\geq 1/2$.

2. It can be shown that $\lim_{t\to\infty} \Pr[H_{t-1}] = 1/2$.

Qed

Theorem B (Algorithm B is good):

Let MC be a Markov Chain. Let the sequence generated by MC be used as input to algorithm B. Then:

1. The output of B is an independent unbiased (possibly finite) sequence of Hs and Ts.

2. If the MC does not enter a deterministic loop (in which

there exits from every state a single arrow with associated probability = 1), then the output of B is infinitely long with probability 1.

3. If all probabilities attached to arrows are 1/2, then B is expected to generate an (independent unbiased) output bit for every 4 input bits.

Proof: 1 and 2 are immediate from the Main Lemma, below. 3 follows for algorithm B as it does for algorithm vN (wherein, for example, 00110110 (8 bits input) causes B to output TH (2 bits)).

Qed

Definitions: Let MC be a Markov Chain. Let $\sigma_1\sigma_2 \cdots \sigma_t$ be a finite sequence of 0s and 1s generated by MC. For each state S_j of MC, let $\sigma_{j_1}\sigma_{j_2} \cdots$ be the subsequence of $\sigma_1\sigma_2 \cdots \sigma_t$ consisting of those symbols generated by MC when it exits state S_j. Call $\sigma_{j_1}\sigma_{j_2} \cdots$ the *subsequence assigned* by $\sigma_1\sigma_2 \cdots \sigma_t$ to S_j. $\sigma_{j_1}\sigma_{j_2} \cdots$ are the *exits* (of $\sigma_1 \cdots \sigma_t$) out of S_j. The *entrances* (of $\sigma_1 \cdots \sigma_t$) into S_j are the symbols of $\sigma_1 \cdots \sigma_t$ that take the MC *into* S_j (from any state). Note that for $S_j \notin \{S_1, S_T\}$, the number of entrances to S_j equals the number of exits from S_j.

Now suppose that two successive symbols σ_{i_k}, $\sigma_{i_{k+1}}$ in the subsequence assigned to S_i are interchanged (all other symbols assigned to S_i and all subsequences assigned to S_j, all j $\neq$ i, being left untouched). Let $\sigma'_{j_1}\sigma'_{j_2} \cdots$ denote the resulting subsequence assigned to S_j. The subsequences associated with S_j, all j, *define* a new sequence $\sigma'_1\sigma'_2 \cdots$ in a natural way: $\sigma'_1 =_{def}$ the first exit (σ'_{1_1}) out of S_1. Inductively, if σ'_j takes MC into some state, S_I, then $\sigma'_{j+1} =$ first exit out of S_I that has not been taken by any of $\sigma'_1 \cdots \sigma'_j$. The map $\sigma_1 \cdots \sigma_t \rightarrow \sigma'_1 \cdots \sigma'_{t'}$ defined (above) by the exchange of σ_{i_k} with $\sigma_{i_{k+1}}$ is called the *order-preserving transformation* of $\sigma_1 \cdots \sigma_t$ into $\sigma'_1 \cdots \sigma'_{t'}$ (see figure 1).

How is the sequence $\sigma'_1 \cdots \sigma'_{t'}$ related to $\sigma_1 \cdots \sigma_t$? For every entrance σ_i of $\sigma_1 \cdots \sigma_t$ into a state there is an exit σ_{i+1} out of that state, except for the last entrance σ_t into S_T. Therefore, $\sigma'_1 \cdots \sigma'_{t'}$ will use a subset of arrows of $\sigma_1 \cdots \sigma_t$! It will *not* necessarily use all of them (see figure 2)!

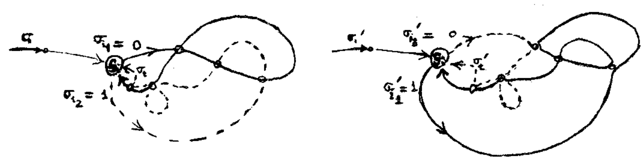

figure 1: an order-preserving transformation that exchanges σ_{i_1} with σ_{i_2}.

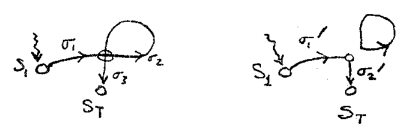

figure 2: interchange σ_2 with σ_3

Lemma 1:

Let MC be a Markov Chain. Let S_i be a state of MC. Let k $\geq$ 1. Let $\sigma_1 \cdots \sigma_{i_k} \cdots \sigma_{i_{k+1}} \cdots \sigma_{i_{k+2}-1}$ be a sequence of 0s and 1s generated by MC, where $i_{k+1} \leq i_{k+2}-1$.

An order-preserving transformation that exchanges σ_{i_k} with $\sigma_{i_{k+1}}$ determines a (unique) sequence $\sigma'_1 \cdots \sigma'_{i_k} \cdots \sigma'_{i_{k+1}} \cdots \sigma'_{i_{k+2}-1}$ with the following properties:

1. $\sigma_1 \cdots \sigma_{i_{k+2}-1}$ and $\sigma'_1 \cdots \sigma'_{i_{k+2}-1}$ both use the same multiset of exit arrows, and therefore both have the same *length* and the same *probability*.

2. $\sigma_1 \cdots \sigma_{i_{k+2}-1}$ and $\sigma'_1 \cdots \sigma'_{i_{k+2}-1}$ both leave the MC with the same state-memory (both sequences store the same state-bit in a state: see definition in algorithm B).

3. $\sigma_1 \cdots \sigma_{i_{k+2}-2}$ and $\sigma'_1 \cdots \sigma'_{i_{k+2}-2}$ (note last index of the two sequences!) both cause algorithm B to output the same number of Hs and the same number of Ts (in possibly different orders). Moreover, $\sigma_{i_{k+2}-1}$ causes B to output H (T) if and only if $\sigma'_{i_{k+2}-1}$ causes B to output T (H).

4. If $\sigma_1' \cdots \sigma'_{i_{k+2^r}-1}$ is transformed into $\sigma_1'' \cdots \sigma''_{i_{k+2^r}-1}$ in an order-preserving transformation that exchanges σ'_{i_k} with $\sigma'_{i_{k+1}}$, then $\sigma_1'' \cdots \sigma''_{i_{k+2^r}-1} = \sigma_1 \cdots \sigma_{i_{k+2^r}-1}$.

Proof: True if $\sigma_{i_k} = \sigma_{i_{k+1}}$. Now assume without loss of generality that $\sigma_{i_k} = 0$ and $\sigma_{i_{k+1}} = 1$.

Suppose $\sigma_1 \cdots \sigma_t$, $t = i_{k+1}-1$, is transformed into $\sigma_1' \cdots \sigma_{t'}'$. We must show that $\sigma_1' \cdots \sigma_{t'}'$ uses *all* the arrows of $\sigma_1 \cdots \sigma_t$ (it uses a subset of those arrows, cf. paragraph preceding Lemma 1).

Suppose not. Then after the interchange, $\sigma_1' \cdots \sigma_{t'}'$ uses a proper subset of arrows of $\sigma_1 \cdots \sigma_t$. The arrows of $\sigma_1 \cdots \sigma_t$ not used by $\sigma_1' \cdots \sigma_{t'}'$, shown dotted in figure 3, form cycles: this is because $\sigma_1' \cdots \sigma_{t'}'$ ends in S_t, so the number of entrances to S_j, any j, equals the number of exits from S_j in both the original sequence $\sigma_{i_k} \cdots \sigma_{i_{k+2^r}-1}$ and the new one $\sigma'_{i_k} \cdots \sigma'_{t'}$.

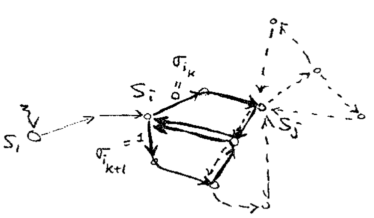

figure 3

Clearly, none of the cycles goes through S_i.

We are to show that there are actually *no* cycles. Consider the first edge, e, of any cycle that is taken by $\sigma_1 \cdots \sigma_t$. Suppose this edge is out of S_j. Suppose there are exactly c cycle edges out of S_j. These are the *last* c edges taken by $\sigma_1 \cdots \sigma_t$ (since they were not taken by $\sigma_1' \cdots \sigma_{t'}'$). After taking edge e, c-1 edges remain to be taken out of S_j by $\sigma_1 \cdots \sigma_t$. But at least c edges must still be taken into S_j by $\sigma_1 \cdots \sigma_t$, which is a contradiction.

It now follows that:

1. We have just proved it.

2. Since every state of MC (except S_i) is exited in the same order by $\sigma_1' \cdots \sigma'_{i_{k+2^r}-1}$ as by $\sigma_1 \cdots \sigma_{i_{k+2^r}-1}$, and since both sequences leave S_i with the same state-bit, λ, the two sequences leave MC with the same state-memory.

3. The sequence $\sigma_1 \cdots \sigma_{i_{k+2^r}-2}$ generates the same number of Hs and same number of Ts as $\sigma_1' \cdots \sigma'_{i_{k+2^r}-2}$. Same argument as for 2 above.

4. This follows because $\sigma'_{i_k} = \sigma_{i_{k+1}}$ and $\sigma'_{i_{k+1}} = \sigma_{i_k}$.

Qed

Algorithm B takes as input a MC and an *infinite* sequence $\sigma_1 \cdots$. If, however, B is fed MC and a finite (rather than infinite) initial sequence $\sigma_1 \cdots \sigma_k$, the output of B will still be a well-defined finite initial string of Hs and Ts. We call $\sigma_1 \cdots \sigma_k$ the finite sequence (of 0s and 1s) that *underlies* that given sequence of Hs and Ts, provided σ_k *caused* the last H or T to be output, i.e., σ_k took MC into a state whose state-bit was previously set to (that last) H or T.

The probability that MC causes B to generate a certain finite initial sequence of Hs and Ts is $\sum \text{prob}(\sigma_1 \cdots \sigma_k)$, where the sum is taken over all strings $\sigma_1 \cdots \sigma_k$ that underlie the given sequence of Hs and Ts.

The idea behind the proof of Theorem B is this: show that pr[TTHH] = pr[HHHT], say, by constructing a 1-1 correspondence between S_{TTHH}, the 0-1 strings underlying TTHH, and S_{HHHT}, the 0-1 strings underlying HHHT. If a string of 0 and/or 1 arrows is used to generate TTHH, then the *same* 0 and/or 1 arrows is used in a different order to generate HHHT.

Main Lemma:

Let MC be a Markov Chain. Let n be any positive integer. Let $S_{T_1T_2 \cdots T_n}$, $S_{T_1T_2 \cdots H_n}$, ... , $S_{H_1H_2 \cdots H_n}$ be 2^n sets, each indexed by a distinct n-bit string of Hs and Ts, each defined to be the set of all finite sequences of 0s and 1s that underlie the indexed sequence of Hs and Ts.

Then there is an equivalence relation $\equiv$ on the set of all strings $\sigma_1 \cdots \sigma_k$ that underlie finite sequences of Hs and Ts, such that:

1. If $\sigma_1 \cdots \sigma_k$ underlies an n-bit sequence of Hs and Ts then every n-bit sequence of Hs and Ts has exactly one underlying sequence $\tau_1 \cdots \tau_l$ that is equivalent to $\sigma_1 \cdots \sigma_k$; moreover, no other sequences are equivalent to $\sigma_1 \cdots \sigma_k$.

2. If $\sigma_1 \cdots \sigma_k \equiv \tau_1 \cdots \tau_l$ then:

   i) $\sigma_1 \cdots \sigma_k$ uses the same multiset of arrows as $\tau_1 \cdots \tau_l$ (consequently $k = l$, $\mathrm{pr}[\sigma_1 \cdots \sigma_k] = \mathrm{pr}[\tau_1 \cdots \tau_k]$, and both $\sigma_1 \cdots \sigma_k$ and $\tau_1 \cdots \tau_k$ take MC into the same state).

   ii) If $\sigma_1 \cdots \sigma_k \equiv \tau_1 \cdots \tau_k$ then $\sigma_1 \cdots \sigma_k$ leaves the MC with exactly the same state-memory as $\tau_1 \cdots \tau_k$.

Proof of Main Lemma: We define $\equiv$ recursively, proving by induction on n that it has the desired properties.

n=1: Define $\equiv$ on $S_T \bigcup S_H$ by setting $\sigma_1 \cdots \sigma_{i_1} \cdots \sigma_{i_2} \cdots \sigma_{i_3 - 1}$ $\equiv \sigma'_1 \cdots \sigma'_{i_1} \cdots \sigma'_{i_2} \cdots \sigma'_{i_3-1}$, where the latter is obtained by an order-preserving transformation that exchanges σ_{i_1} with σ_{i_2} (so $\sigma'_{i_1} = \sigma_{i_2}$ and $\sigma'_{i_2} = \sigma_{i_1}$). Then, by Lemma 1, $\equiv$ satisfies 1 and 2 (of the Main Lemma) on $S_T \bigcup S_H$.

General n: Now assume $\equiv$ has been defined so that it satisfies 1 and 2 on the set of all finite sequences that underlie m-bit strings of Hs and Ts, for all m $<$ n. We extend $\equiv$ as follows: Let $\sigma_1 \cdots \sigma_j \cdots \sigma_k \in S_{T_1 H_2 \cdots T_{n-1} T_n}$, say, one of the 2^n subsets whose index terminates in T. Delete symbols from the right hand side of $\sigma_1 \cdots \sigma_j \cdots \sigma_k$ to get the sequence $\sigma_1 \cdots \sigma_j$ of 0s and 1s that underlies the n-1 bit string $T_1 H_2 \cdots T_{n-1}$. If $\sigma_1 \cdots \sigma_j \equiv \sigma'_1 \cdots \sigma'_j$ then, by Lemma 1, both strings leave the MC in the same vector-state. Extend $\equiv$ to: $\sigma_1 \cdots \sigma_j \cdots \sigma_k \equiv \sigma'_1 \cdots \sigma'_j \cdots \sigma_k$, where the additional σs on both sides are the *unprimed* σs that were previously deleted. In this way, $\equiv$ is extended to an equivalence relation on the union of all 2^{n-1} subsets $S_{\cdots T_n}$ whose indices *end* in T_n. In similar fashion, extend $\equiv$ to an equivalence relation on the union of all 2^{n-1} subsets $S_{\cdots H_n}$ whose indices end in H_n. Finally, extend the equivalence relation by setting each string in the *particular* set $S_{T_1 T_2 \cdots T_{n-1} T_n}$ (whose index consists of n Ts) equivalent to the unique string in $S_{T_1 T_2 \cdots T_{n-1} H_n}$ gotten by applying the order-preserving transformation of Lemma 1. Note that Lemma 1, part 3, assures that strings in $S_{T_1 T_2 \cdots T_{n-1} T_n}$ are transformed into strings in $S_{T_1 T_2 \cdots T_{n-1} H_n}$. The result is an equivalence relation satisfying 1 and 2 on the set of all finite sequences that underlie strings of up to n Hs and Ts.

Qed

4. How to generate an independent unbiased sequence of Hs and Ts from an unknown Markov chain.

While algorithms vN, A and B appear practical, algorithms C and D below do not. There is one exception: a properly modified algorithm C may be practical in case MC is known but its start state is unknown.

Let k or k(n) denote the number of n-state finite state machines (FSM). Let the i^{th} n-state FSM, M_i, have states q^i_j, $j \in \{1, \cdots, n\}$, where q^i_1 = start state. Define *M(n)*, the *universal simulator of n-state FSMs*, or *universal FSM* for short, to be the FSM with n^k states, each denoted by $< q^1_{j_1}, \cdots, q^i_{j_i}, \cdots, q^k_{j_k} >$ for $j_1, \cdots, j_k \in \{1, \cdots, n\}$. M(n) has start state $< q^1_1, \cdots, q^i_1, \cdots, q^k_1 >$. If a state q^i_j of M_i maps under 0 to q^i_l, then associated state $< \cdots q^i_j \cdots >$ of MC(n) maps under 0 to $< \cdots q^i_l \cdots >$.

Algorithm C:

Input: A positive integer n; a sequence of 0s and 1s (produced by an n-state MC).

Output: A sequence of Hs and Ts (independent and unbiased).

Begin:

Apply Algorithm B to input M(n) and the given sequence of 0s and 1s.

End.

Lemma:

Let MC_i be an n-state MC with underlying FSM M_i (M_i gotten from MC_i by deleting the probabilities on the arrows). Let pr$[0|q_j^i]$ = probability attached to the 0-arrow out of state q_j^i of MC_i. Attach probabilities to the arrows of the universal FSM, M(n), to get a Markov chain MC(n) that simulates MC_i, by setting pr$[0|< \cdots q_j^i \cdots >]$:= pr$[0|q_j^i]$.

Then MC(n) is a Markov chain having the same input/output conditional probabilities (eg., pr[1 | 0010]) as M_i.

Theorem C:

Algorithm C on being input a pair (n; an infinite sequence of 0s and 1s produced by an n-state Markov chain, MC_i) will output an independent unbiased sequence of Hs and Ts. If MC_i does not enter a periodic cycle, then with probability 1 the output of algorithm C will be infinitely long.

After an initial delay, algorithm C generates bits in expected linear time: the expected number of output bits per input bit is $\sum_{i=1}^{k} \frac{pr[<S_i>]}{p_i q_i}$, where k is the number of n-state FSMs, $pr[<S_i>]$ = probability that MC(n) (see above Lemma) is in state $<S_i>$, and $p_i = 1 - q_i =$ probability assigned to the 1-arrow out of $<S_i>$.

Algorithm D:

Input: A sequence of 0s and 1s (produced by a MC with a finite but unknown number of states).

Output: A sequence of Hs and Ts (eventually independent and unbiased).

Begin:

1. Set f(1) := 0.
2. **For** n = 1, 2, 3, ...

   do: Apply Algorithm C to input (n; $\sigma_{f(n)+1}\sigma_{f(n)+2} \cdots$) until a H or T is produced. Then

   (i) Output that H or T, and

   (ii) Set f(n+1) := index i such that σ_i caused above H or T to be produced.

   od

End

Theorem D:

Algorithm D on being input an infinite sequence of 0s and 1s produced by a finite state Markov Chain, MC, (no bound given on the number of states in MC) will output a sequence of Hs and Ts that is eventually (after some point) independent and unbiased. If MC does not enter a periodic cycle, then with probability 1 the output of algorithm D will be infinitely long.

5. Conclusions

A variety of sources (e.g., back-biased zener diodes) can be used to generate truly random (as opposed to pseudo-random) sequences. This is important, for example, for producing the seeds required by pseudo-random number generators.

Unfortunately, the bits produced by physical sources appear to not be completely random; rather, the probability of each bit is conditional on the preceding bits. Producing *completely random,* i.e., independent and unbiased, bits is difficult primarily because of this dependence. This paper studies how to generate independent unbiased sequences of Hs and Ts from certain correlated sources, finite state Markov chains. The study reveals a surprising and counterintuitive finding: a simple algorithm that one would expect would produce independent and unbiased bits from a Markov source is shown to fail. This algorithm A is a natural extension of von Neumann's algorithm for generating unbiased Hs or Ts from a biased coin.

Though algorithm A is faulty, a small (but nonobvious) modification of it yields a correct algorithm B. The resulting B differs from A primarily in the time when it elects to output bits. B uses only slightly more memory than A, is just as frugal as A in its use of Markov chain bits, and outputs completely random Hs and Ts in expected linear time.

Theorems A, B, C, D can be extended to the more usual definition of Markov chain in terms of matrices (n arrows out of each state, one to each of the n states; each arrow labelled by the state it goes to rather than by 0 or 1; probabilities attached to all arrows out of a state sum to 1). Theorems A, B, C can be extended to the problem of generating Hs and Ts in expected linear time with an efficiency that very nearly achieves the entropy of the source [Elias].

Acknowledgements:

Michael Fischer and his student, Josh Cohen, are doing interesting work along related lines: they consider how to generate sequences that are as independent and unbiased as possible, given that an output *must* be generated for every 100 bits input from a biased coin.

Miklos Santha and Umesh Vazirani have their own novel approach: they generate n-bit-long random sequences that appear independent and unbiased to any (not necessarily computable) statistical test, when the underlying source is a collection of $\frac{1}{\delta}\cdot(\log n)\cdot(\log^* n)$ independent parallel sources of random bits. Each source is unknown and arbitrary except that its conditional probability of generating a 1 bit at time t (whatever the history of that bit) lies in the interval $[\delta, 1-\delta]$, for some fixed $\delta > 0$.

In addition to the above four, I want to thank David Blackwell, David Gifford, John Gill, Michael Saks, and Avi Wigderson for many valuable discussions.

The students in Berkeley course CS 276, Number Theory and Cryptography, suffered through the initial discovery and presentation of the material in this paper. I would like to thank each of them for their participation in this venture!

Bibliography:

1. Cohen, Josh D. (1983), "How and When to Flip a Biased Coin Fairly," Draft paper, Comp. Science Dept., Yale U., 13 pp.

2. Elias, Peter (1972), "The Efficient Construction of an Unbiased Random Sequence," Ann. Math. Statist. Vol 43, No. 3, 865-870.

The following paper has a comprehensive bibliography:

3. Stout, Quentin F. and Warren, Bette (1984), "Tree Algorithms for Unbiased Coin Tossing with a Biased Coin," Ann. Prob. Vol. 12, No. 1, 212-222.

4. von Neumann, John (1951), "Various Techniques Use in Connection with Random Digits," Notes by G. E. Forsythe, National Bureau of Standards, Applied Math Series, Vol. 12, 36-38. Reprinted in von Neumann's Collected Works, Vol. 5, Pergamon Press (1963), 768-770.