

1. Introduction

Programming by example (pbe) is the inference of programs from samples of their behavior. Rather than have users program a task, they need only specify examples of how to perform it, from which a program is synthesized that accomplishes the task itself. Pbe schemes have been proposed for office information systems (e.g. Halbert, 1984; MacDonald & Witten, 1987), operating system interaction (e.g. Waterman *et al.*, 1986), robot programming (e.g. Andreae, 1984; Heise & MacDonald, 1989), graphical editing (e.g. Myers, 1988; Maulsby *et al.*, 1989), and text editing (Mo & Witten, 1990; Nix, 1984).

The dominant paradigm in pbe is the procedural approach, where a sequential trace of user actions is generalized into an explicitly procedural representation. One alternative is the transformational method espoused by Nix (1984), where sequential information in the user's trace is ignored and a functional mapping is sought between input and output. Another is a knowledge-intensive approach where domain knowledge and plan inference techniques combine to infer the user's goals so that they can be accomplished automatically.

In this paper we address the question of evaluating alternative procedural descriptions of a sequential trace of user actions. Moreover, we are concerned with the case in which the procedure is developed incrementally, on the fly, rather than being inferred from a completed action trace. Experiments have shown that this incremental strategy helps to reduce noise and free variation in user action sequences that would otherwise confound attempts to infer an underlying procedure (Maulsby *et al.*, 1990). By working incrementally the system is able to suggest actions at the earliest possible opportunity, which has the triple effect of helping the user early on, reducing user errors, and encouraging consistency in situations that admit several acceptable variants of an action sequence—thereby reducing the complexity of induction.

We work with procedures that are incompletely specified so that, in effect, some actions are non-deterministic. In practice non-determinism will be resolved by the user: the pbe system will suggest the most likely action and the user will either accept it, ask for an alternative, or enter the correct action. While the aim of pbe is to infer a completely specified deterministic procedure that predicts all actions uniquely and correctly, it is necessary to work with nondeterministic procedures for two reasons. First, the information needed to disambiguate alternative courses of action may simply not be available to the program—for example, it might depend on unexpressed goals of the user or on unpredictable real-world events. Second, even if the critical information is available, in the early stages of operation the system will not have had enough experience to sift it from the mass of potentially relevant facts, yet still may be able to offer predictions that are useful even though not always correct.

2. Goal

In this paper it is assumed that procedures are represented as finite-state automata. The pbe system operates in a continual cycle as follows:

- predict the next user action, if possible;
- receive the next user action, whether in the form of an accepted prediction or as a new action constructed by the user;
- update the model of the program to reflect that action, if necessary.

Although not explicitly mentioned, we take it for granted that each of these steps will be conditioned on the current application-specific state or “situation,” as provided by the system in which programming-by-example takes place.

In the case where the program does not predict the correct action, the third step will modify its structure to accommodate that action in future. In general, there will be several plausible ways in which it can be modified, none of which can be immediately

determined to be superior to the others. This leads to the strategy of maintaining a *set* of plausible models, and elaborating the third step above to:

- modify each member of the plausible model set to reflect that action, if necessary;
- select that model currently judged best, as the basis for the next prediction.

It is likely that the modify operation adds new members to the model set—indeed, each member may spawn several offspring. The model set must then be continually pruned to prevent exponential growth.

The goal of this research is to devise a rational method for selecting the “best” model from a set of candidates. This will be used for prediction as dictated by the final step above, and perhaps also for pruning the model set to implement a beam-search exploration of the space of possible models.

We are *not* concerned here with the question of how to modify individual models to conform to a new, unpredicted, action. While this operation forms an essential part of the incremental pbe methodology, in our experience it is very easy to come up with plausible heuristics for modifying models. We have experimented with many different heuristics, ranging from context-based ones where each action is predicted on the basis of a small number of predecessors, to programming ones that force models to conform to accepted programming practice—for instance, no branching out of loops.

It seems unlikely that there exists a single, “best,” approach to model formation and modification. Some problems lend themselves naturally to a production-system-style model as a set of quasi-independent situation-action “rules.” Others are strictly sequential and are best implemented as a state machine. The notion of “state” may need to be abstracted from the sequence of actions and/or situations. Creating suitable models is an interesting problem, but, for general pbe, will almost certainly remain one for which no optimal solution exists.

In summary, a generate-and-test approach to model formation seems appropriate. This paper addresses the testing part.

3. Examples

Some specific illustrations of the problem will prove helpful. To help focus on the question of model evaluation, the examples are presented as sequence-identification problems. They are nevertheless representative of actual pbe problems, and to add verisimilitude the final example is taken from a real pbe system which can be seen in the accompanying videotape.

EXAMPLE 1. Figure 1 shows two alternative models for sequences that contain an a, two b's, one c, and one or more d's, i.e. $(abbc d^+ \bullet)^+$. The symbol \bullet is used as a terminator, and superscript $+$ means repetition one or more times. The question is, when is it rational for the system to prefer the second model, that encodes the information that exactly two b's appear, to the first, that treats consecutive b's the same as consecutive d's? Presumably not after Sequence 1 of Figure 1, but perhaps after Sequence 2.

EXAMPLE 2. Figure 2 shows another problem with two competing models, one of which captures the "true" structure of the sequence while the other does not. How long must the sequence run before the second model, larger but more accurate, is preferred to the first?

EXAMPLE 3. Figure 3 shows a sequence that, at first sight, looks like random coin-tosses. (It is taken from an anecdote of Andreae's, 1977, about a rigged penny-tossing machine.) Closer examination shows some deterministic structure—when taken in threes (starting after the second H of the sequence shown), the third member of each triple is a copy of its predecessor. The random model can be expressed as a 2-state

automaton, while the “true” one below is much more complex, requiring 6 states. How much of the sequence must be seen before the larger model is preferred? Another large model that partially captures the structure is shown on the right: should this ever be the preferred model?

EXAMPLE 4. Figure 4a shows a program formed by the Metamouse system for pbe in a graphical environment, taken from Maulsby *et al.* (1990). In fact, the system has been taught to align and sort a set of boxes, and the actions do accomplish this task. However, the program is not completely autonomous, since it requires prodding from the user at one point; a more accurate program is shown in Figure 4c. (The reader is referred to the videotape for a graphic demonstration of the problem.) Our question is, how many repetitions of the procedure are required before the larger but more predictive model is preferred?

4. Proposed evaluation measure

We seek a suitable metric for evaluating a candidate model with respect to a given sequence. The measure we propose is

- the number of bits needed to transmit both sequence and model using a coding scheme capable of economically transmitting any sequence and model of it.

This is a form of the “minimum description length” principle proposed by Rissanen (1985) and applied by a number of authors to machine learning problems. However, it is by no means a routine application of this principle, because, as we shall see, coding the sequence and model is best done using a novel incremental coding technique. The actual coding method is necessarily a heuristic one—the question of *optimal* coding of a sequence into a minimum number of bits is well known to be undecidable (Chaitin, 1974).

The central hypothesis of the paper is that

- this measure corresponds with intuition about when it is worth using a more complex model to describe a sequence.

In other words, when does the evidence from the sequence “justify” a larger model? For example, in Figure 1 the first sequence certainly does not justify preferring the second model to the first, while the second probably does. This process of induction is an intuitive one and a good metric will agree with our own judgement.

We now define a particular coding scheme capable of economically transmitting any sequence along with a model of that sequence. This is done in several steps, illustrated in Figure 5. First, we dispense with the tedious and irrelevant business of actually producing a bit-string that represents the result of coding and replace it with an entropy measure that gives the size of the resulting bit-string without actually producing it. Second, the entropy of a sequence with respect to a given probabilistic model is defined. Third, this definition is extended to models in which the probabilities are not specified, by associating counts derived from the sequence itself with the model’s transitions. Fourth, it is shown how entropy can be decreased by manipulating the counts as the sequence is processed. Fifth, it is shown how almost the same result can be obtained without transmitting the counts at all. Finally, this method of “adaptive transmission” is extended to transmit the model structure as well.

DISPENSING WITH THE CODING OPERATION. There is no need to perform the coding operation because it is known how to code a symbol whose probability is known to be p in $-\log p$ bits. The method of “arithmetic coding” has the property that

- it is able to code a symbol with probability p in a number of bits arbitrarily close to $-\log p$;
- the symbol probabilities may be different at each step.

(See Witten *et al.*, 1987, for further discussion, and an implementation.) Consequently we need only discuss coding schemes in terms of their entropy. If a sequence of

symbols $a_1 a_2 \dots a_n$ can be predicted with probabilities p_1, p_2, \dots, p_n , then the entropy of the sequence is

$$\sum_{i=1}^n -\log p_i \text{ bits.}$$

ENTROPY W.R.T. A PROBABILISTIC MODEL. Given a sequence of symbols and an automaton that represents it and has probabilities attached to the transitions, the entropy of the sequence with respect to the model is defined in the usual way:

$$\sum -c_{ij} \log p_{ij} \text{ bits,}$$

where c_{ij} is the number of times that transition ij is traversed, p_{ij} is the probability attached to it, and the summation is over all transitions in the model. It is assumed that the model is one that “accepts” the sequence, in the sense that appropriate transitions exist for it to be possible to trace the sequence through the model, and that it is unambiguous, in the sense that transitions out of a state always lead to states with different labels. Figure 5a shows such a model for the example sequence, and the counts derived from it; the entropy of the sequence with respect to the probabilistic model is 6.83 bits.

One could imagine first transmitting a probabilistic model of a sequence, then sending the sequence with respect to it. However, this is infeasible because probabilities are, in general, fractional numbers that take a good deal of space to encode to any reasonable accuracy.

SELF-ENTROPY W.R.T. A NON-DETERMINISTIC MODEL. A non-deterministic model of a sequence is a probabilistic model which has no probabilities assigned to the transitions, like that of Figure 5b. Given a sequence of symbols and a non-deterministic model of it, the entropy of the sequence with respect to the model is defined by counting the number of times each transition is traversed when the sequence is passed through the model and then forming probabilities from these counts. The self-entropy is

$$\sum_i [c_i \log c_i - \sum_j c_{ij} \log c_{ij}] \text{ bits, where } c_i = \sum_j c_{ij}.$$

This is obtained by simply replacing p_{ij} in the previous expression by the empirically observed frequency c_{ij}/c_i . It can easily be shown that these particular values of p_{ij} will minimize the entropy. For example, the non-deterministic model of Figure 5b gives a self-entropy of 6.75 bits.

It would be feasible to transmit (a) a non-deterministic model of a sequence, then (b) the count associated with each transition, then (c) the sequence with respect to the model. One simple way of transmitting the model is to specify, for each of n states, the destinations of the q transitions that could emanate from that state, where q is the size of the alphabet used. This requires $q \log n$ bits per state, or $nq \log n$ bits in total. For the example model this amounts to 24 bits.

To transmit the counts, note that for an N -symbol sequence and a model with r transitions, there are $N C_r$ ways of assigning counts to transitions, and so if nothing is known a priori about the distribution of counts, transmitting them will consume

$$\log \frac{(N+r-1)!}{N!(r-1)!} \text{ bits.}$$

This amounts to 10.97 bits in the example. In fact, as N and r grow one can show that this enumerative method is (asymptotically) no better than transmitting each count c_{ij} individually, which can be done (asymptotically) in $\log c_{ij}$ bits.

However, given the model and counts, a more economical coding of the sequence is possible.

SELF-ENTROPY WITH DECREASING COUNTS. Given transition counts that are known to correspond to a particular sequence, the count on a transition should be decremented each time it is taken. For example, if a particular state has two exits, taken 2 times and 6 times respectively for a particular sequence, then the probability is 1/4 that the first exit will be taken. Once it has been taken, however, the probability should drop to 1/7 since

it will be taken just once more (and the state will be visited 7 times more). The last time the state is visited, the exits will have counts of 0 and 1 (or vice versa), and it will be *known* which will be taken. Figure 5c shows the calculation for the example sequence, based on the non-deterministic model of Figure 5b.

This is a better way to use the counts than the previous one. However, both methods share the problem that both model and counts must be transmitted.

SELF-ENTROPY WITH INCREASING COUNTS. It is possible to avoid transmitting the counts explicitly. Clearly the entropy value will be the same whether the counts begin at their maximum value and decrease to zero, or begin at zero and increase to their maximum value. But in the latter case they need not be transmitted at all (except, perhaps, the total, to terminate the sequence)! There is a catch, however. When they are zero the model can make no predictions, and hence the sequence cannot be transmitted. An obvious solution is to start all counts at 1 rather than 0. Coding performance on the sequence itself is reduced, since the probabilities are less accurate, but this is more than outweighed by the fact that the counts need not be transmitted. Figure 5d shows the calculation, again based on the model of Figure 5b.

There is no theoretical justification for starting the counts at 1, rather than, say, 2—or even 1.5. This is a variant on the so-called “zero-frequency problem” that, in the absence of any information, there can be no rationale for probability estimation (see Witten & Bell, in press, for a fuller discussion). In practice, coding is relatively insensitive to the particular solution that is chosen.

In fact, when we come to adaptive transmission of the model *structure*, it will be beneficial to use a different solution to the zero-frequency problem. Call transitions that have not yet occurred “novel,” and for each state record the frequency of novel events—the number of transitions emanating from the state—and the frequency of non-novel events—the total count on the transitions. This provides a basis for estimating the

probability that the next event will be novel, called the “escape probability.” If it is novel, a different mechanism must be used to specify which of the zero-frequency transitions has occurred.

ADAPTIVE TRANSMISSION OF THE MODEL STRUCTURE. So far it has been assumed that the model structure is transmitted in advance. However, the way is now paved for incremental transmission of structure as well as counts.

At each stage, a symbol will either

- cause an existing transition to be followed out of the current state;
- cause a new transition to be created to an existing state;
- cause a new state to be created, with a transition to it.

In the first case, the coding probability is $c_{ij}/(c_i+1)$ to transmit the fact that the j th transition occurred. In the second case, an escape code is generated with probability $1/(c_i+1)$, the fact that an existing state is used is transmitted in 1 bit, and the identity of that state is transmitted in $\log n$ bits where n is the current number of states in the model. In the third case, the escape is generated as before; a code indicating new-state is transmitted in 1 bit, and the symbol associated with the new state is transmitted in $\log q$ bits where, as before, q is the size of the alphabet used.

This is the coding method used; its operation on the example sequence is illustrated in Figure 6. The result is that the sequence and model are coded together in just 21.51 bits, significantly less than the other methods discussed. Some small optimizations can be applied—for example, when q transitions emanate from a state, one can use the coding probability c_{ij}/c_i instead of $c_{ij}/(c_i+1)$, since escape can never occur—but these have negligible effect on the results below.

5. Performance on the examples

We now apply the evaluation measure developed above and illustrated in Figure 6 to the examples of Section 3 to see whether it accords with our intuitive judgement as to when there is enough information to justify more complex models.

EXAMPLE 1. The first example sequence of Figure 1 and the first model are transmitted together in 21.87 bits, while that sequence and the second model require 22.52 bits—thus the first model is preferred. When the number of d's in the sequence is varied, this difference remains constant at 0.64 bits. However, the second example sequence favors the second, more accurate, model (38.35 bits) over the first (40.76 bits); the difference again remains constant (at 2.41 bits) regardless of the number of d's in the three subsequences. In fact, the second model is preferred whenever the sequence consists of two or more subsequences of the form $abbd^+a$.

EXAMPLE 2. Results for the second example are shown graphically in Figure 7. The smaller, simpler model has a smaller evaluation measure, and is therefore preferred, until the first 14 elements of the sequence have been seen, which comprise two repetitions of the form a^+ba^+c . The models remain neck and neck until 22 elements have been seen. Once a total of just under 4 repetitions of a^+ba^+c have been seen, the second model is uniformly preferred to the first.

EXAMPLE 3. The coin-tossing example is interesting because people rarely spot the structure themselves. The true model (6 states) is *much* more complex than the naive one (2 states), and many examples must pass before sufficient evidence has accumulated for it to be preferred by our evaluation measure. Figure 8 shows that the simple model remains superior until just over 60 symbols have been seen, and the two remain neck and neck for another 40 symbols. In fact, although it is not shown on the

graph, the more complex model remains superior from symbol 103 onwards. It is interesting to note that the third model, which predicts the sequence better than the naive one (but worse than the true model), is never preferred over the naive model during the example sequence because its predictive superiority does not justify the added complexity.

EXAMPLE 4. Both of the models shown in Figure 4 were tested on the action trace generated by sorting 4 boxes, and the simpler (but inferior) model of Figure 4a was preferred by the evaluation measure (by about 4 bits). However, when a second action trace, generated by sorting 3 boxes, was appended the correct model of Figure 4b was preferred (by 6 bits). When a third action trace was appended the correct model was preferred by a larger margin (9 bits).

6. Conclusions

The entropy-based measure developed here is a principled way of evaluating alternative procedural descriptions of a sequential trace of user actions. Based on the minimum description length principle, it differs from existing applications of that principle in that it operates adaptively, building the final description incrementally. This appears to be the most economical way of coding a sequence along with a model that accepts it, and indeed there is some theoretical justification for the approach (Cleary & Witten, 1984).

The use of the evaluation measure presupposes some means of generating alternative candidate models. That is not addressed in the paper. However, we have found it easy to come up with ways of generating models: the problem is choosing between alternative possibilities.

The test of the measure is whether its preference of one procedure over another, based on a particular example sequence, accords with human judgement. Examples were presented that illustrate different aspects of procedures (unrolling loops, creating

states, detecting complex statistical anomalies) and here the measure performs well. A final example illustrated its use in a real programming-by-example system to detect an ill-formed procedure.

Acknowledgements

We gratefully acknowledge the stimulating research environment provided by the Knowledge Science Lab at the University of Calgary. This research is supported by the Natural Sciences and Engineering Research Council of Canada and by Apple Computer Inc.

References

- Andreae, J.H. (1977) *Thinking with the teachable machine*. Academic Press, London.
- Andreae, P.M. (1984) "Constraint limited generalization: acquiring procedures from examples," *Proc. American Association of Artificial Intelligence National Conference*, Austin, Texas; August.
- Chaitin, G.J. (1974) "Information-theoretic computational complexity," *IEEE Trans Information Theory IT-20* (1): 10-15; January.
- Cleary, J.G. and Witten, I.H. (1984) "A comparison of enumerative and adaptive codes," *IEEE Trans Information Theory IT-30*(2): 306-315; March
- Halbert, D. (1984) "Programming by example." Research Report OSD-T8402, Xerox PARC, Palo Alto, California.
- Heise, R. and MacDonald, B.A. (1989) "Robots acquiring tasks from examples" *Proc ITESM 2nd International Symposium on AI*, Monterrey, Mexico.

- MacDonald, B. A. & Witten, I. H. (1987) "Programming computer controlled systems by non-experts," *Proceedings of the IEEE SMC Annual Conference*, 432–437. Alexandria, Virginia.
- Maulsby, D.L., Kittlitz, K.A., & Witten, I.H. (1989) "Metamouse: specifying graphical procedures by example," *Proceedings of ACM SIGGRAPH*, 127–136. Boston, Massachusetts.
- Maulsby, D.L., Witten, I.H., Kittlitz, K.A. and Franceschin, V.G. (1990) "Inferring graphical procedures: the complete Metamouse," Research Report, Department of Computer Science, University of Calgary, Canada.
- Mo, D.H. and Witten, I.H. (1990) "Learning text editing tasks from examples: a procedural approach," Research Report, Department of Computer Science, University of Calgary, Canada.
- Myers, B.A. (1988) *Creating user interfaces by demonstration*. Academic Press.
- Nix, R. (1984) "Editing by example," *Proc. ACM Symposium on Principles of Programming Languages*: 186–195. Salt Lake City, Utah; January.
- Rissanen, J. (1985) "Minimum description length principle," in *Encyclopaedia of statistical sciences, Vol. 5*, edited by E S.Kotz and N.L.Johnson, pp. 523–527. Wiley, New York.
- Waterman, D., Faught, W., Klahr, P., Rosenschein, S. and Wesson, R. (1986) "Exemplary programming: applications and design considerations." In *Expert systems: techniques, tools and applications*, edited by P. Klahr and D. Waterman, pp. 273–309. Addison-Wesley.
- Witten, I.H., Neal, R., Cleary, J.G. (1987) "Arithmetic coding for data compression," *Communications of the ACM* 30(6): 520–540; June.

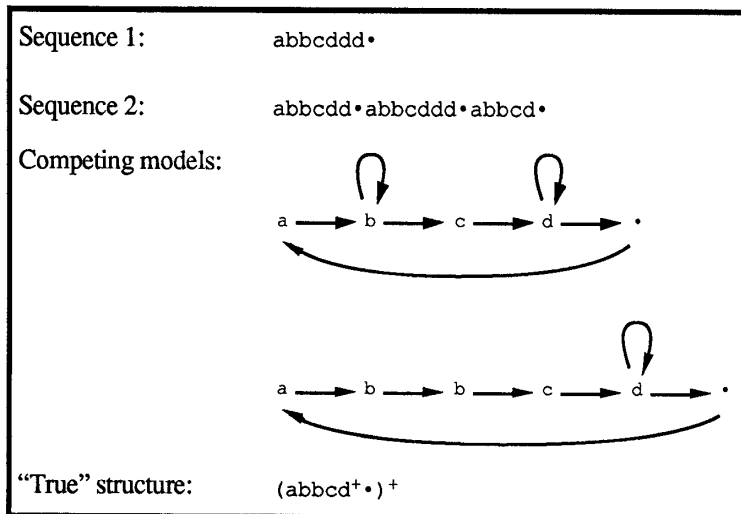


Figure 1 How much evidence is required to unroll a loop?

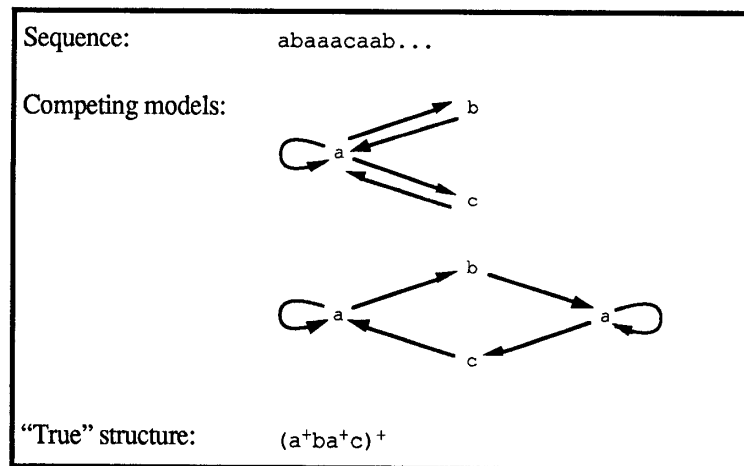


Figure 2 How much evidence is required to create a state?

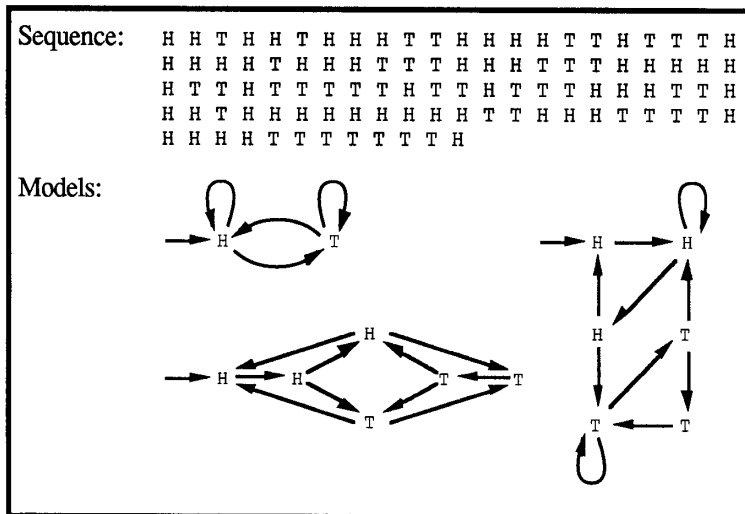
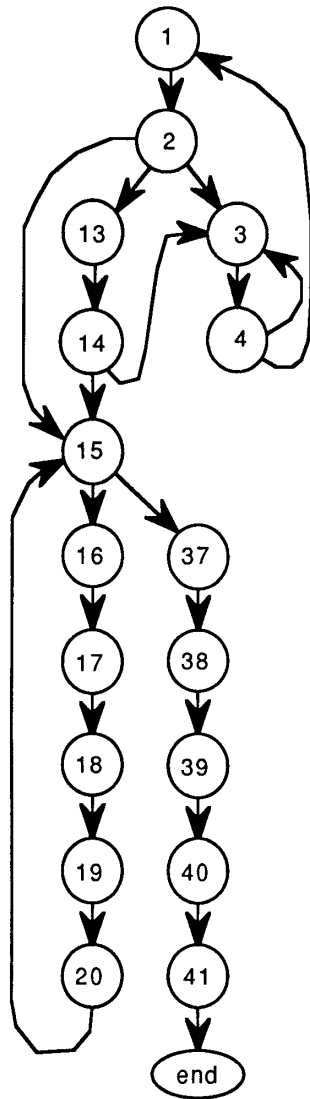


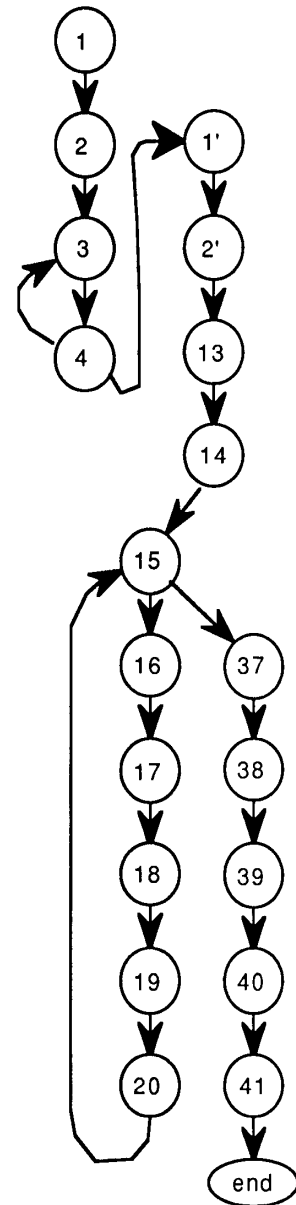
Figure 3 How much evidence is required to see that the coin is unfair?



(a)

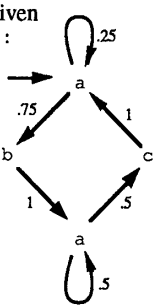
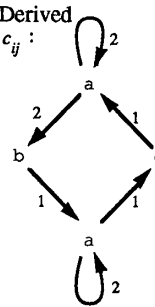
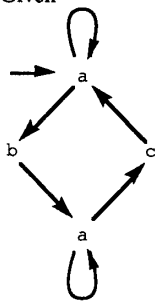
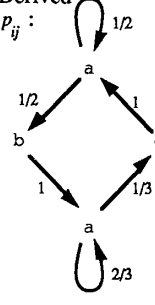
1. Move to position (ask-user)
2. Draw-line Base- / Gapline to position (ask-user)
3. Move to grasp (Box.center)
4. Drag Box to touch (Box.bottom : Baseline.?)
- 5-12. Repeat 3-4 three times
Then do 1-2 again
13. Move to position (constant)
14. Draw-line Sweepline to position (constant)
15. Move to grasp (Sweepline.?)
16. Drag Sweepline to touch (Box.top : Sweepline.?)
17. Move to grasp (Box.center)
18. Drag Box to touch (Box.left : Gapline.right)
19. Move to grasp (Gapline.midpt)
20. Drag to touch (Gapline.left : Box.right)
37. Delete Sweepline
38. Move to grasp (Gapline.midpt)
39. Delete Gapline
40. Move to grasp (Baseline.?)
41. Delete Baseline

(b)



(c)

Figure 4 (a) Program formed by Metamouse
(b) Explanation of action in each state
(c) Well-structured program for the same task

Sequence: abaaacaab		Bits			
Model		To send sequence	To send statistics	To send structure	
Entropy (w.r.t. a given probabilistic model)	Given p_{ij} : 	Derived c_{ij} : 	6.83	Large (real numbers)	24
Self-entropy (w.r.t. a given non- deterministic model)	Given 	Derived p_{ij} : 	6.75	10.97	24
Self-entropy (with decreasing counts)	a b a a a c a a b $\frac{2}{4}$ $\frac{2}{3}$ $\frac{1}{1}$ $\frac{2}{3}$ $\frac{1}{2}$ $\frac{1}{1}$ $\frac{1}{1}$ $\frac{1}{2}$ $\frac{1}{1}$	4.17	10.97	24	
Self-entropy (with increasing counts)	a b a a a c a a b $\frac{1}{2}$ $\frac{1}{3}$ $\frac{1}{1}$ $\frac{1}{2}$ $\frac{2}{3}$ $\frac{1}{4}$ $\frac{1}{1}$ $\frac{2}{4}$ $\frac{2}{5}$	8.49	0	24	
Self-entropy (including transmission of model)	See Figure 6	Total: 21.51			

6.83

Large
(real
numbers)

24



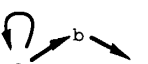
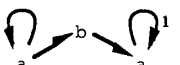
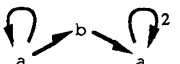
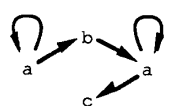
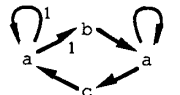
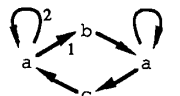
Model so far	Upcoming symbol	Action	Probability	Bits
a	a	Don't create a new state	$\frac{1}{2}$	1
	b	Don't follow existing transition ($c_i=1$) Create a new state Assign it a symbol from {a,b,c}	$\frac{1}{2}$ $\frac{1}{2}$ $\frac{1}{3}$	$2 + \log 3$
	a	Create a new state Assign it a symbol from {a,b,c}	$\frac{1}{2}$ $\frac{1}{3}$	$1 + \log 3$
	a	Don't create a new state Identify next state from {1,2,3}	$\frac{1}{2}$ $\frac{1}{3}$	$1 + \log 3$
	a	Follow existing transition ($c_{ij}=1, c_i=1$)	$\frac{1}{2}$	1
	c	Don't follow existing transition ($c_i=2$) Create a new state Assign it a symbol from {a,b,c}	$\frac{1}{3}$ $\frac{1}{2}$ $\frac{1}{3}$	$1 + 2 \log 3$
	a	Don't create a new state Identify next state from {1,2,3,4}	$\frac{1}{2}$ $\frac{1}{4}$	3
	a	Follow existing transition ($c_{ij}=1, c_i=2$)	$\frac{1}{3}$	$\log 3$
	b	Follow existing transition ($c_{ij}=1, c_i=3$)	$\frac{1}{4}$	2
Total:				21.51

Figure 6 Example use of the proposed evaluation measure

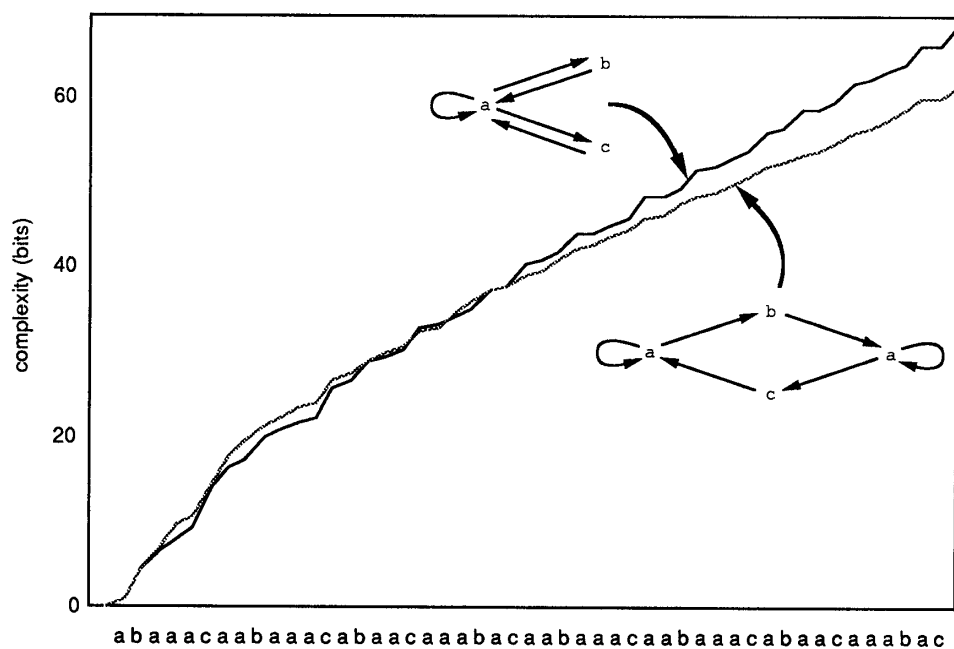


Figure 7 Evaluation measure versus sequence length for the example of Figure 2

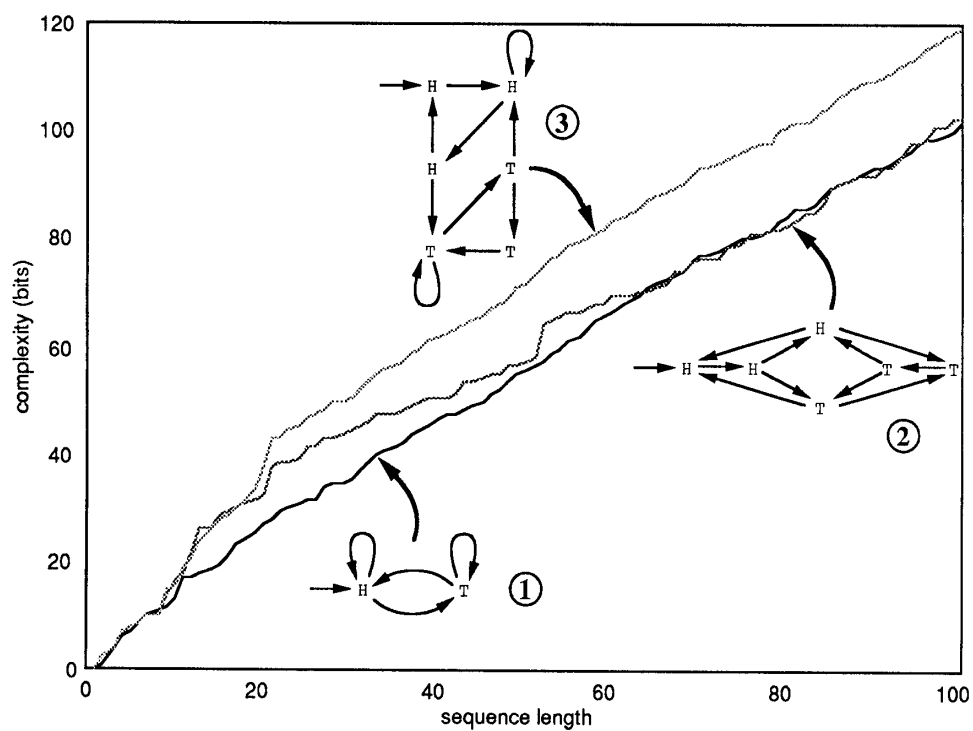


Figure 8 Evaluation measure versus sequence length for the example of Figure 3