ENGINEERING APPLICATIONS OF FINITE AUTOMATA

Bernard E. Lutter and Ralph C. Huntsinger
Chemical Engineering Department
South Dakota School of Mines and Technology
Rapid City, South Dakota

ABSTRACT

Probabilistic models provide a mechanism for computer simulation of a wide variety of engineering processes. This paper compares several techniques, including Markov Chains, Turing Machines and Simulated Evolution for modeling deterministic and stochastic processes. The results form the basis for choosing that class of finite automata which best represents the process being modeled.

Fortran IV digital computer programs written include programs which generate Turing Machines, first and second order Markov Chains and Evolutionary Machines.

INTRODUCTION

In recent years the study of automata has been acquiring increasing importance for engineers in many fields. For some time, the capabilities of these automata, computation-wise, have been of the greatest interest to logicians and mathematicians, including some modern linguists and biologists who search for models of the as yet mysterious structures of living organisms. The expanding literature on the use of finite automata as probabilistic models, however, demonstrates growing interest in the application of these mechanisms to engineering phenomona. This paper explains several modeling techniques using Turing Machines, Markov Chains, and Simulated Evolution. Some results of simulation experiments are presented in order to illustrate the capabilities and limitations of each.

THE MARKOV PROPERTY AND TURING MACHINES

Observed data is said to exhibit Markov properties if preceding events have some influence on succeeding events. For instance, observed data may show some pattern or repetitive cycle which may or may not be readily apparent. Consider for example, the following finitie set of seqential events:

A, B, C, D

As an aid to understanding the mechanism by which these events occur and as a means of predicting future events, it is often helpful to construct a process model, especially when no simple mathematical model is known.

One of the simplest process models which may be used for this purpose is called a Turing Machine and is defined in terms of a finite automation G (P, Q, R, λ), where

P = imput alphabet
Q = set of states
R = output alphabet
λ = output function

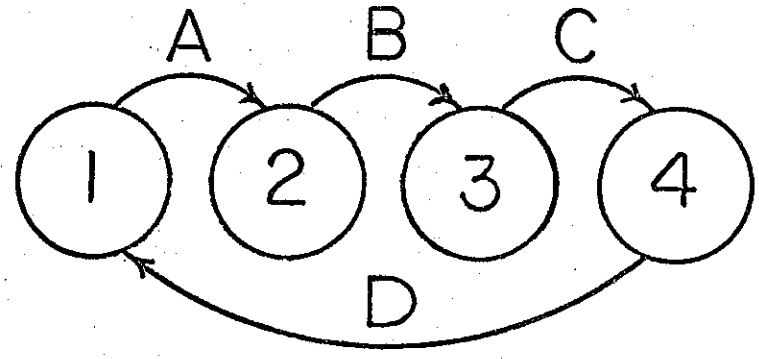

Figure 1. Graphic representation of a Turing Machine.

More simply, an event is uniquely determined by the preceding event, either observed or predicted. As each succeeding event is predicted, the corresponding next-state transition is made, and in this way, a series of future events may be determined. This is accomplished by simply modeling a finite set of observations.[1]

The set of events mentioned earlier are represented as a Turing Machine in Figure 1. Starting in state 1, the event A, is predicted and the transition to state 2 takes place. If this process is continued, the future events, A, B, C, D, A, B, etc., are determined.

The limitations of the Turing Machine become readily apparent with its application to more complex processes. This discussion, however, serves to illustrate the basic ideas of finite automata and to show the development of more general algorithms.

STOCHASTIC PROCESSES AND MARKOV CHAINS

Most engineering processes do not exhibit an exactly reproducible output. Such processes cannot be modeled adequately with a strictly deterministic model such as the Turing Machine. A probabilistic, or stochastic process may be defined as "some possible actual, e.g. physical, process in the real world, that has some random or stochastic element involved in its structure."(2)

[1]A more complete definition may be found in di Forino (1).

Markov Chains belong to the large class of stochastic models. The Markov model differs from the deterministic model in that the state of the process at time T_n or at event E_n is statistically related to the state at T_{n-1} or event E_{n-1}. The degree of dependency of a given state upon previous states, commonly termed the "memory," is high for the deterministic process, lower for the Markov process and is equal to zero for a purely random process. Stochastic process models, in general, exhibit varying degrees of dependence, or memory, according to the process in question. A first order Markov Chain, for instance, does not "remember" as much as Markov Chains of higher order.

Perhaps the Markov Chain could be best illustrated with an example. Suppose it is desired to model the variation in the output water temperature from an industrial cooling tower for the purpose of prediction and control. This temperature is mainly affected by changes in the weather. Since changes in atmospheric conditions are generally considered stochastic, a stochastic process model such as the Markov Chain may be applied. Figure 2 shows a set of hourly temperature readings generated by a cooling tower simulation program[1] using actual weather data collected over a period of 8 days. The total variation is arbitrarily divided into four equal temperature ranges labeled 1, 2, 3, and 4 corresponding to 20, 40, 60, and 80 degrees respectively. A sequence of temperature readings (state transitions) is then generated as shown at the bottom of Figure 2.

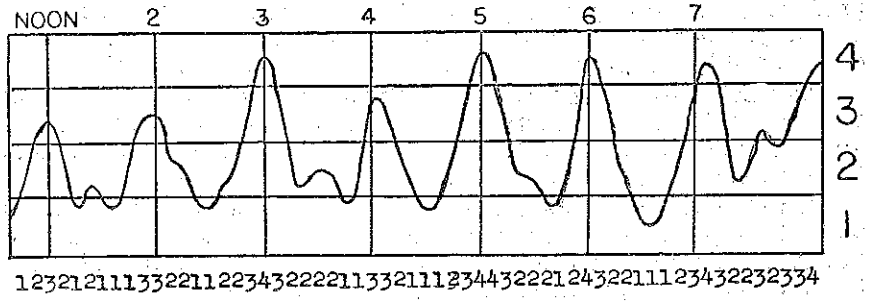

Figure 2. Variation in output water temperature of cooling tower over an 8-day period.

The simulation of the temperature cycle using a Markov Chain involves the use of a transition probability matrix (Table I). This matrix is defined in terms of the transitions from each state to its successor. Starting at the left, the first transition is from 1 to 2.

[1]The Abstract of a paper describing the simulation can be found in Lutter and O'Connor (3).

This is indicated by a tally mark in the transition matrix according to the following rule: the rows of the matrix represent the given state and the columns represent the state to which the transition proceeds. Thus, the first transition is indicated by a tally mark in row 1 of column 2, and the second transition, from 2 to 3, is recorded in row 2 of column 3. This procedure is repeated for a specified period (8 days in this case) or may be carried out during operation of the cooling tower. The resulting tally matrix is shown in Table 1-a. To obtain the corresponding transition probabilities for each

Table I. Tally and Transition Matrices

	1	2	3	4
1	𝍸 𝍸 ||||	𝍸 |||	||	
2	𝍸 ||	𝍸 ||	𝍸 |	|
3		𝍸 |||	|||	||||
4			||||	|

	1	2	3	4
1	.473	.421	.106	0
2	.333	.333	.286	.048
3	0	.533	.200	.267
4	0	0	.800	.200

row, the number of tally marks for each possible transition is divided by the total number in that row. The Transition Probability Matrix is shown in Table 1-b.

Once the Transition Probability Matrix is obtained, it is possible to simulate the stochastic variation of temperature. To do this, one starts in a given state and draws random numbers to select the succeeding transitions. Starting in state 1, the probability of remaining in state one is slightly greater than going to state 2, and four times as great as that of going to state 3. The chances are greater, then, that the model will begin with the transition 1-1 or 1-2 rather than 1-3 or 1-4. A typical model generated in this manner is shown in Figure 4-b.

A better representation of the temperature variation may be obtained by dividing the total variation into a greater number of segments, or by decreasing the transition interval. As the transition interval becomes very small, however, the probability of a transition from one state to itself approaches unity. On the other hand, if the interval used is too large, some transitions may be missed entirely. Thus, it is evident that some judgment is needed regarding the size of interval which will accurately describe the process. Transition and state (temperature) intervals should be chosen which will minimize both repetition and discontinuities in the state array.

For some processes this model (a first order Markov Chain) would be entirely adequate. If the temperature variation is known to be more deterministic (less random) than that represented by the first order model, a model with more "memory" can be used. A second order Markov Chain uses the two preceding states as a criterion for making the next transition. A second order model of the temperature variation in Figure 2 is shown by the transition probability matrix in Figure 3.

A problem is often encountered when a transition is predicted for which no actual observation has yet been made. In this event, the corresponding row of the Transition Probability Matrix contains nothing but zeroes. In order to make an intelligent decision, it has been found necessary to compute a lower order TPM as a "back-up" for use when this problem occurs. This scheme is also helpful in starting the simulation when only one starting state is given.

LEARNING SYSTEMS AND THE EVOLUTIONARY MODEL

The third modeling technique considered is usually termed a self-adaptive, or learning system. This class of automata is typified by Neural Nets and other such biological representations which are generally said to exhibit artificial intelligence. In a sense, the Markov Chain possesses learning capabilities. As more process data is observed, appropriate changes are seen in the Transition Probability Matrix. However, this is an indirect type of learning. It tends to average each new observation with the combined effect of all past observations. Of course, this type of learning has application to many processes, especially those represented by finite sets of data. When this technique is applied to continuous data however, such as the digitized temperature data presented in the previous example, some problems are encountered. For example, one would normally expect to see a high temperature at approximately noon each day. As seen in Figure 4, this is not the case with the Markov models. The overall pattern is present but due to the probabilistic nature of the predictions, the exact location of individual cycles is somewhat confused. Thus, a modeling technique which would account for this problem must possess pattern recognition capabilities.

The basic concepts of evolutionary programming described by Fogel, Owens, and Walsh (4) have been applied to this type of data. Fogel's scheme simulates the natural process of evolution to create a decision making entity composed of a finite no. of states interconnected by a complex logic network. Specifically, this finite-state machine is expressed as a quintuple (A, B, S, λ, δ, S_0) where A, B, and S are finite sets (A being the input set, B being the set of outputs, and S being the set of internal states), λ is a function from A x S into S, δ is a function from A x S into B and S_0 is a member of S termed the initial or start state (5).

	1	2	3	4
1 1	.30	.40	.30	0
1 2	.17	.17	.50	.16
1 3	0	0	1.0	0
1 3	0	0	0	0
2 1	.72	.28	0	0
2 2	.40	.40	.20	0
2 3	0	.33	.17	.50
2 4	0	0	1.0	0
3 1	0	0	0	0
3 2	.33	.56	.11	0
3 3	0	.67	0	.33
3 4	0	0	.75	.25
4 1	0	0	0	0
4 2	0	0	0	0
4 3	0	1.0	0	0
4 4	0	0	1.0	0

Figure 3. Second order Transition Matrix

A finite-state machine may be viewed as a type of Turing machine which can modify itself using simulated evolution to produce a model which best represents the process on the basis of the evidence at hand. This is accomplished by arbitrarily choosing an

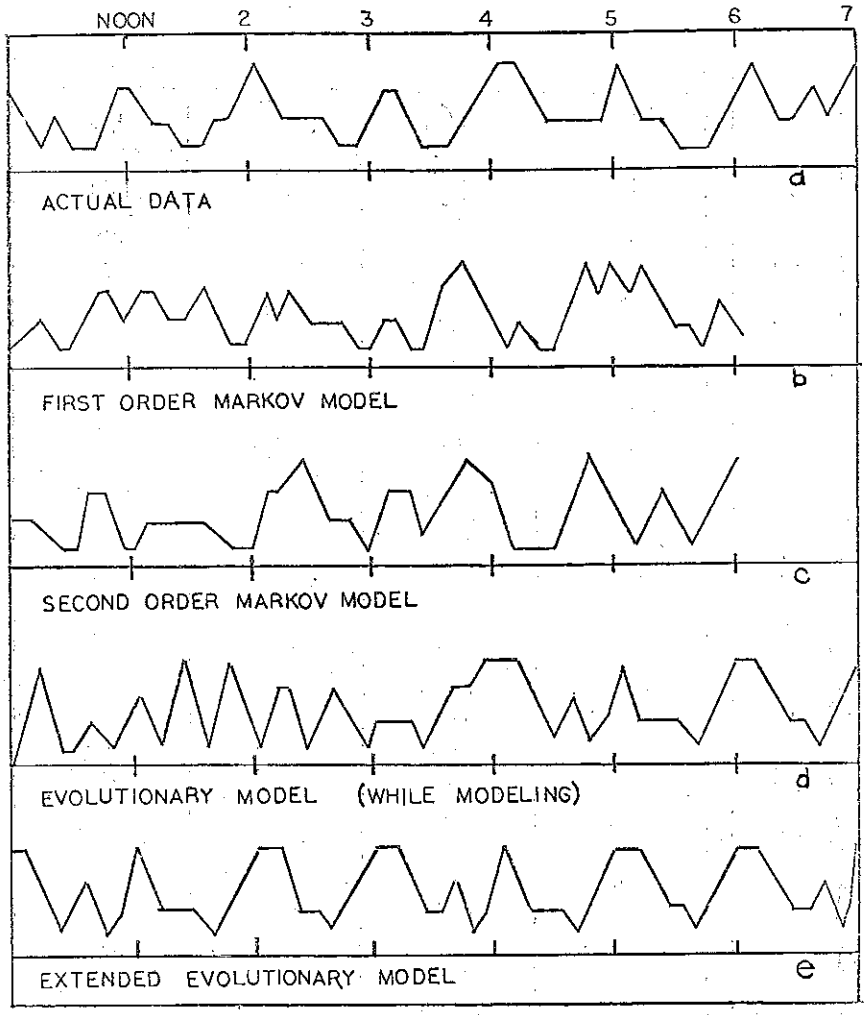

Figure 4. Digitized Temperature Profiles

initial machine and a starting state and evaluating this machine over the set of process output data (termed the "recall") thus-far observed. For example, consider the arbitrary 3 state machine pictured in Figure 5. To evaluate this model in terms of the temperature data presented earlier, an initial state is chosen. In this case, state A was arbitrarily chosen. The first temperature observed was that represented by the symbol 1.

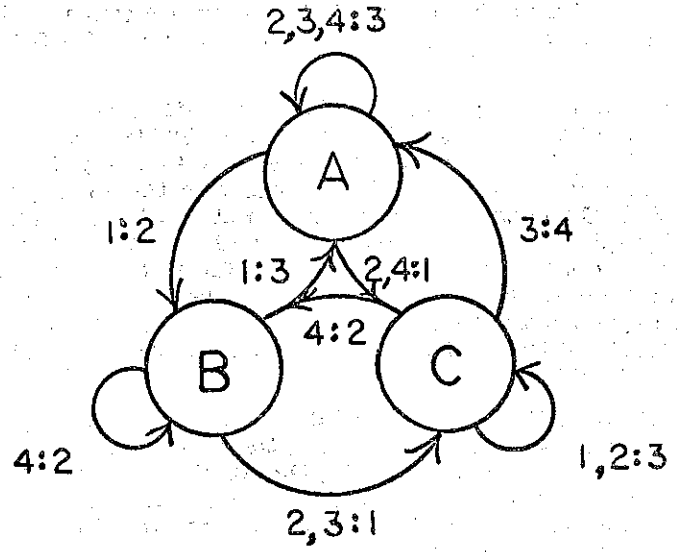

Figure 5

According to the logic of the machine as it is presently constructed, an input of 1 indicates a transition to state B, shown by the arrow from state A, and an output at 2, shown to the right of the colon. This output symbol represents a prediction of the next process output (temperature) symbol to be observed. The next symbol observed was actually a 2, therefore the correct symbol was predicted. To evaluate this model in terms of its ability to predict each successive symbol in the recall, an error cost matrix is defined and is shown in Table 2-b, no error cost is given for a correct prediction of the next symbol. Error costs for wrong predictions may be defined in terms of the relative degree of error involved in making that prediction.

Table 2-a. Evaluation of model over recall Table 2-b.

PRESENT STATE	A	B	C	A	C
INPUT	1	2	3	2	1
NEXT STATE	B	C	A	C	
OUTPUT	2	1	4	1	
ERROR COST	0	2	1	0	

EVALUATION

	1	2	3	4
1	0	1	2	3
2	1	0	1	2
3	2	1	0	1
4	2	1	1	0

COST MATRIX

For example, if a symbol 1 is predicted when a 4 was actually seen, this mistake is assigned a higher error cost than if a 2 had been predicted.

Thus, the error cost for the first evaluation was zero and is shown at the bottom of the first column in Table 2-a. Continuing the evaluation, the present state is now B and the input is now the 2 which was actually observed. According to the model, the next state is state C and the predicted next input is a 1. The next symbol is actually a 3 and the error cost for this mistake is 2. The accumulated error cost for the evaluation thus far is (0 + 2) = 2. This procedure is continued over the total recall length and the accumulated error cost is divided by the number of observations to give the relative worth of the model. For the first four predictions, the relative worth is 0.75.

Evolution of the model now takes place in an attempt to improve its worth as a representation of the process. Four changes in the model (mutations) are possible; the number of states, the starting state, a state transition, or a prediction. One mutation is made and the model is again evaluated over the recall. If the error cost is lower, the model is retained; if not, it is rejected. In either case, the search for a better model continues until a decision (prediction of unobserved process data) is required.

The simulation of this modeling technique requires the use of two additional matrices; the Next State Matrix, and the Output Matrix. The machine in Figure 5 is defined by the matrices, shown in Table 3 and can be evaluated using the error cost matrix shown in Table 2-b.

TABLE 3.

INPUT

STATE	1	2	3	4
A	B	A	A	A
B	A	C	C	B
C	C	C	A	B

NEXT STATE

INPUT

STATE	1	2	3	4
A	2	3	3	3
B	3	1	1	2
C	3	3	4	2

OUTPUT

Evolution is simulated with random numbers to choose a specific mutation thereby causing an appropriate change in one of these matrices. This process continues until ultimately, a perfect (or nearly perfect) machine is evolved.

RESULTS OF MODELING EXPERIMENTS

Figure 4 shows the results obtained using first and second order Markov Chains and the Evolutionary technique to model the given temperature data. The limitations of the Markov models concerning pattern recognition were mentioned earlier. This is shown by the errors in the relative position of highs in the first two models. The evolutionary model, however, is not limited in this respect. The learning capabilities of this technique can be seen in Figure 4-d. Recall that the evolutionary model is developed as more actual data is observed. The model shown is totally inadequate in the beginning, but has improved sufficiently after 3 days observations to predict reasonable temperatures. The extended temperature prediction for the next seven days, shown in Figure 4-e, illustrates the logic which was developed after observation of temperature data over the previous seven days. Note that the pattern is accurate and that the extended predictions make up a repetitive three day cycle. These predictions may change, of course, with the observation of more data.

CONCLUSIONS

The Evolutionary modeling technique is superior to those considered earlier in many engineering applications. In modeling continuous systems, the Evolutionary technique has shown pattern recognition capabilities. In addition, it was found that the core storage requirements for the computer simulation using Evolutionary programming were generally lower. The simple structure of the Evolutionary machine and its ability to learn have also made this technique a versatile one. A mathematical investigation conducted by Walsh (5) has indicated a large class of environments which are amenable to representation by these finite-state machines.

This is not to say that the Evolutionary technique is the answer to every technical modeling problem. Certainly an understanding of the process in question is essential in choosing a probabilistic or deterministic model. If sufficient knowledge of the process exists, there is no reason to use a probabilistic model at all. On the other hand, when little or nothing is known concerning the process, probabilistic models provide a point of entry which, hopefully, will lead to a better understanding and control of the underlying process that gives rise to the observed phenomena.

REFERENCES

1. di Forino, A. C., "Generalized Markov Algorithms and Automata," Automata Theory, E. R. Caianiello, ed., p. 116 (1966).

2. Bartlett, M. S., An Introduction to Stochastic Processes With Reference to Methods and Applications, The University Press, Cambridge, p. 1 (1960).

3. Lutter, B. E. and O'Connor, P. J., A Digital Computer Application of A Graphical Solution to Cooling Tower Design, Proc. S. Dak. Acad. Sci., **46,** 260 (1967).

4. Fogel, L. J., Owens, A. J., and Walsh, M. J., Artificial Intelligence Through Simulated Evolution, John Wylie and Sons, Inc., New York (1966).

5. Walsh, M. J., Evolution of Finite Automata for Prediction, Technical Report No. RADC-TR-67 555, Rome Air Development Center, Griffis AFB, New York (1967).

6. Carne, E. B., Artificial Intelligence Techniques, Macmillan and Company, Ltd., London (1965).

7. Krumbein, W. C., Fortran IV Computer Programs for Markov Chain Experiments in Geology, University of Kansas, Lawrence (1967).

8. Feigenbaum, E. A. and Feldman, J., Computers and Thought, McGraw-Hill Book Co., New York (1963).

9. Nelson, R. J., Basic Concepts of Automata Theory, Proc. 20th Nat. Conf. Association for Computing Machinery, p. 138 (1965).

10. Caianiello, E. R., ed., Automata Theory, Academic Press, London (1966).