DECISION SCIENCE INC.

AD 714201

PREDICTION AND CONTROL THROUGH THE USE OF AUTOMATA AND THEIR EVOLUTION

Reproduced by
NATIONAL TECHNICAL
INFORMATION SERVICE
Springfield, Va. 22151

DDC
RECEIVED
NOV 13 1970
B

DECISION SCIENCE, INC.
4508 MISSION BAY DRIVE
SAN DIEGO, CALIFORNIA
92109 (714) 273-2922

PREDICTION AND CONTROL THROUGH THE USE OF AUTOMATA AND THEIR EVOLUTION

Final Report On:
Contract No. N00014-66-C-0284

Prepared By:
Michael J. Walsh
George H. Burgin
Lawrence J. Fogel, Principal Investigator

September 15, 1970

DDC
RECEIVED
NOV 13 1970
B

This [illegible] and sale; its distribution is unlimited.

INTRODUCTION

Finite-state machines provide a natural means for representing the logic which may underlie a sequence of data derived from a sensed environment or for depicting the transduction between stimulus and response of such an environment. Such representation permits expansion of the logic in terms of arbitrary input and output languages so long as these are expressed within finite alphabets. Further, the machines may be of arbitrary specificity so long as they have only a finite number of states. Thus, no unnatural constraint is imposed, as is so often the case when a sequence of data is expressed in terms of a best linear fit or when a transduction is expressed in terms of a linear difference or differential equation. More powerful logical entities can be considered, but these may be identified with problems which reduce their acceptability.

The concept of evolutionary programming was conceived as a means to find a most appropriate finite-state machine for the purpose of prediction or modeling in terms of the available data base and an arbitrary expression of goal (a payoff or error-cost matrix). Formulation of this concept can be traced through various publications. A review of the

concepts of evolutionary programming, its application in terms of prediction experiments, data reduction and analysis, and control system design were described (reference 1), this work being supported in part by the Office of Naval Research. This reference also indicates application in terms of gaming and the possible automation of the scientific method. Further practical and theoretical studies along these lines were described (references 2 through 12). These references concern the prediction of solar flares, the problem of self-referencing, modeling of the human operator, the design of self-sufficient prostheses, and gaming in various contexts.

In this work certain problems have been recognized and resolved while others have been identified and brought into closer perspective. Fundamental to all these is the limit of evolutionary programming in terms of efficiency and the potential use of this technique for gaining greater insight into aspects of natural evolution.

This final report does not contain previously published material. Rather, it includes specific findings which serve to clarify the capability and limitations of using automata and evolutionary programming for the purpose of prediction and control.

DISCUSSION

Choosing the size of a finite-state machine always presents a problem. Obviously any finite sequence of symbols can be perfectly fit by a large enough machine, but it would most likely be a poor predictor, for the states would be exercised so rarely that the machine would not represent any of the underlying regularity of the process whereby the sequence of symbols was generated. This problem was recognized early in the development of evolutionary programming and so a complexity factor was introduced to inhibit the growth of machine size. Initially this factor was input data to the program, but to choose a suitable value for the complexity factor, some a priori knowledge was required (whether it is cyclic, stochastic, and so forth). Various techniques were developed so that the program could directly determine machine size or indirectly control the growth of machines by assigning appropriate values to the complexity factor. However, none of these techniques work well for all of the environments which were considered.

The primary difficulty in this regard was that the choice between a parent and offspring had to be on the basis of fit-score, and larger machines generally tend to have better fit-

scores due to the fact that many of the state-input pairs are exercised only once over the recall. The output algorithm then assigns the symbol giving smallest error-cost to each state-input pair and, thus, assigns the correct symbol to each such singleton state-input pair.

The first technique used to eliminate this overemphasis on fit-score requires that the program maintain several classes of machines, usually three or four. One class consists of a one-state machine while a second class consists of machines having a number of states which exceeds some specified lower limit but remains less than some upper limit determined by the computer memory. The third class or additional classes consists of machines with specified lower and upper bounds on the number of states. The lower bounds usually exceed one and the upper bounds are generally below the lower limit of the class of large machines.

After the initial period during which the machine having the best fit-score is used as the predictor, a machine is chosen from that class which indicates the best "action" score over the last k predictions, k being some preassigned number. In general, this method yielded an improvement over the previous scheme which maintained only a single machine, but it increases the storage problem as well as the cost of computtation. Note that here several machines must be referenced and the current machine of each class must be mutated during each mutation phase.

Another algorithm was then devised which considered the result of only those inputs which exercise other than singleton state-input pairs in determining the average error score (the fit-score). This technique as well as the earlier algorithms were tested against a first-order Markov process (using the single machine algorithm versus the multiple machine algorithm since the latter always had a one-state machine which would be the best finite-state for a first-order Markov process). The newly developed technique demonstrated not only a better prediction score, but through its use, the size of the machines decreased while machine size grew in the older version.

In many situations the plant or other transducer under consideration is actually a composite of smaller logical entities, the composition being parallel, series, or a combination of parallel and series. In such a case the number of states in a single machine representation of the transucer is, in general, the product of the number of states in the smaller machines. For example, parallel compositions of a six-state and a nine-state machine results in a 54-state machine (although the actual number of states required may be less than 54 since there may be some redundancy identifiable and some state combinations may not be reached from the given starting state). Thus, when a machine can be decomposed into smaller

machines, the total number of states which must be considered is considerably reduced. This also raises the question as to whether or not a machine, even if not decomposable, is a submachine of a decomposable machine. Thus, theoretical and experimental effort was applied to the problem of parallel decomposition.

A program was written to evolve a parallel composition of two finite-state machine which would fit the transduction of a larger finite-state machine. In the initial experiment, a six-state machine with eight-symbol input was known to be the parallel composition of a two-state machine (the output symbols of which were the high-order bit of the larger machine's symbols) and a three-state machine (the output symbols of which were the two lower-order bits of the larger machine's symbols). After 385 mutations the program successfully evolved the appropriate two-state and three-state machines.

Before applying the program to arbitrary machines, some criteria should be established as a guide for determining if the program is evolving machines which, in parallel composition, adequately simulate or approximate the given machine. Thus, an effort was made to determine the "best" parallel decomposition of a given machine for a given decomposition of the input-output alphabet of a machine, if possible. This would provide a means for assessing how well the evolutionary program performs. Theoretical aspects of this problem were also considered.

In the case of a machine which is the parallel composition of two machines, a decomposition scheme should produce these two machines. This case was considered first:

Let $M_1 = (A, B_1, Q_1, \lambda_1, \delta_1)$ and

$M_2 = (A, B_2, Q_2, \lambda_2, \delta_2)$ be reduced machines and let

$M = (A, B. Q, \lambda, \delta)$

be the parallel composition of M_1 and M_2 so that $B = B_1 \times B_2$, $Q = Q_1 \times Q_2$, $\lambda(a,(q_1, q_2)) = (\lambda_1(a, q_1), \lambda_2(a, q_2))$ and $\delta(a,(q_1, q_2)) = (\delta_1(a, q_1),\delta_2(a, q_2))$.

Let $M_1' = (A, B_1, Q, \lambda, \delta_1')$ with

$\delta_1' (a, (q_1, q_2)) = \delta_1(a, q_1)$ and

$M_2' = (A, B_2, Q, \lambda, \delta_s')$ with

$\delta_2' (a, (q_1, q_2)) = \delta_2(a, q_2)$.

Then M_1 is the reduced machine of M_1' and M_2 is the reduced machine of M_2'.

With this result as a guide, consider a machine $M = (A, B, Q, \lambda, \delta)$ and any decomposition of B into a direct product of two sets, say, B_1 and B_2, for example, $B = B_1 \times B_2$. Then there exists unique functions δ_1 and δ_2 with δ_1 mapping $A \times Q$ into B_1 and δ_2 mapping $A \times Q$ into B_2 so that $\delta(a, q) =$

$(\delta_1(a, q), \delta_2(a, q))$.

Let $M_1' = (A, B_1, Q, \lambda, \delta_1)$ and

$M_2' = (A, B_2, Q, \lambda, \delta_2)$

Then if under this decomposition of B, M is as before the parallel composition of machines M_1 and M_2, it can be shown that M_1 is the reduced machine of M_1' and M_2 is the reduced machine of M_2'.

In general, if M_1 is the reduced machine of M_1' and M_2 is the reduced machine of M_2', then there is a submachine N of the parallel composition of M_1 and M_2 such that M is the reduced machine of N. This is the case if M is reduced, otherwise M and N have the same reduced machine.

In view of these results, for a given decomposition of the output alphabet, the reduced machines of the machines M_1' and M_2' constructed as above from a given machine M yield the best parallel decomposition of M.

In the parallel case, the machines are essentially independent so that independent modeling is possible. This is not necessarily true in the series composition. If the inputs and outputs of each part is known, then the problem reduces to that of modeling individual plants. If the intermediate outputs and hence the intermediate inputs are not available so that only the initial input and final output are known,

then it is not possible to independently model the parts.

For example, consider the series composition of plant P_1 followed by plant P_2 where only the input to P_1 and the output from P_2 are known. Suppose modeling by the series composition of a finite-state machine M_1 followed by machine M_2 is being attempted. If machine M_1 is kept fixed, then it is possible to evolve the best machine M_2, since input and output to M_1 are not affected by changes in M_2. However, when M_2 is fixed, evolving a best M_1 presents a problem in that a mutation of M_1 and the resulting change in its output changes the input environment to M_2. However, since the expected output from M_2 is invariant, this would, in general, result in a degradation of M_2 as a model. Further, since expected output from M_1 is not known, the current versions of the evolutionary program cannot be applied since this knowledge is required in their operation. It was, therefore, felt that effort should be directed toward a theoretical investigation of series composition to determine, if possible, effective algorithms for evolving the series composition of finite-state machines. Some preliminary results are given here.

Suppose machine M is the series composition of machines M_1 and M_2 (composition in the order named), where

$$M_1 = (A, B, Q, \lambda_1, \delta_1, q_0) \text{ and}$$

$$M_2 = (B, C, P, \lambda_2, \delta_2, p_0).$$

Then:

$$M = \left(A, C, Q \times P, \lambda, \delta, (q_0, p_0)\right) \quad \text{where}$$

$$\lambda\left(a, (q, p)\right) = \left(\lambda_1(a, q), \lambda_2\left(\delta_1(a, q), p\right)\right) \quad \text{and}$$

$$\delta\left(a, (q, p)\right) = \delta_2\left(\delta_1(a, q), p\right) .$$

Observe that $Q \times P = \bigcup_{q \in Q} \left(\{q\} \times P\right)$ and also that

$$Q \times P = \bigcup_{p \in P} \left(Q \times \{p\}\right) .$$

Consider the first of these decompositions of $Q \times P$ and let $S_i = \{q_i\} \times P$ so that $Q \times P = \cup S_i$. Now for $i \neq j$, $S_i \cap S_j = \emptyset$ (null or empty set). Hence, $\{S_i\}$ is a partition of $Q \times P$. Moreover, if $\lambda_1(a, q_i) = q_j$, then

$$\lambda(a, S_i) = \lambda\left(a, \{q_i\} \times P\right)$$

$$= \left(\lambda_1(a, q_i), \lambda_2\left(\delta_1(a, q_i), P\right)\right)$$

$$\subset \{q_j\} \times P$$

That is,

$$\lambda(a, S_i) \subset S_j$$

It is also true that if $T_i = Q \times \{p_i\}$, then $\{T_i\}$ is a partition of $Q \times P$. However, consider (q_1, p_i) and (q_2, p_i) with $q_1 \neq q_2$. Both of those are in T_i.

Now:

$$\lambda\left(a, (q_1, p_i)\right) = \left(\lambda_1 (a, q_1), \lambda_2 \left(\delta_1 (a, q_1), p_i\right)\right) \quad \text{and}$$

$$\lambda\left(a, (q_2, p_i)\right) = \left(\lambda_1 (a, q_2), \lambda_2 \left(\delta_1 (a, q_2), p_i\right)\right)$$

But, in general, it is possible that $\delta_1(a, q_1) \neq \delta_1(a, q_2)$ and hence that $\lambda_2\left(\delta_1(a, q_1), p_i\right) \neq \lambda_2\left(\delta_1(a, q_2), p_i\right)$. It is, therefore, not always the case that $\lambda(a, T_i) \subset T_j$ for some $_j$. So while $\{T_i\}$ is a partition, it is not invariant under the λ mapping while the partition $\{S_i\}$ is.

On the basis of the partition $\{S_i\}$, it follows that for a finite-state machine to be a series composition of two machines, it is necessary that a partition of the states exists which is invariant under the λ mapping. The set of singleton states forms such a partition but this would lead to the trivial series composition of the given finite-state machine with a one-state machine. This partition is excluded as an allowable one, for example, in an allowable partition, not all of the sets in the partition can be singleton sets.

With the partition $\{q_i \times P\}$ as a guide, the problem of decomposing a given finite-state machine into a series compo-

sition of machines is considered. Let M be a finite-state machine where $M = (A, C, S, \lambda, \delta, s_0)$ and suppose $S = S_1 \cup \cdots \cup S_k$ such that for each a in A and each i there is a j such that $\lambda(a, S_i) \subset S_j$. (For a set X, let $|X|$ be the cardinality of X, that is, the number of elements in it). If necessary, relabel the states so that

$$S_i = \left\{ s_{ij} \;\middle|\; 1 \leq j \leq |S_i| \right\}$$

and $s_0 = s_{11}$. Let I_k be the set consisting of the first k positive integers, let $B = A \times I_k$ and denote the pair (a_i, j) by b_{ij}. Let

$$Q = \left\{ q_i \;\middle|\; i \text{ in } I_k \right\}$$

be a set of k new symbols and consider the finite-state machine M_1 with

$M_1 = (A, B, Q, \lambda_1, \delta_1, q_1)$ where

$\lambda_1(a, q_i) = q_j$ whenever

$\lambda(a, S_i) \subset S_j$ and $\delta_1(a_i, q_j) = b_{ij}$. Let

$$P = \left\{ p_i \;\middle|\; 1 \leq i \leq r \right\}$$

be a set of r new symbols, where $r = \max|S_j|$, let c_o be in C, and consider the finite-state machine M_2 with

$$M_2 = (B, C, P, \lambda_2, \delta_2, p_1) \quad \text{where}$$

$$\lambda_2(b_{ij}, k_n) = p_m \quad \text{whenever} \quad n \leq |S_j| \quad \text{and}$$

$$\lambda(a_i, s_{jn}) = s_{km}$$

$$= p_r \text{ otherwise and}$$

$$\lambda_2(b_{ij}, p_n) = \delta(a_i, s_{jn}) \quad \text{whenever} \quad n \leq |Sj|$$

$$= c_o \text{ otherwise.}$$

Theorem. M is equivalent to a submachine of the series composition of M_1 and M_2. If $|S_j| = r$ for all j, M is equivalent to the series composition of M_1 and M_2.

The identification problem arises whenever it is required to determine a logical representation for the transduction which is presumed to couple two sequences of symbols (the presumed input and output of a black box). Evolutionary programming was demonstrated as a means for addressing this problem.

For example, the finite-state machines shown in Figures 1 and 2 were driven by a forcing function, these input and output sequences being used as a basis for modeling by both the evolutionary program and the human operator. Specifically, Tables 1 and 2 indicate the input and output sequences in each of these experiments. Table 3 provides a comparison of the modeling capability of the human operator with that of the evolutionary program in terms of predicting each next symbol on the basis of the currently held model.

The identification problem often arises within the context of prior information concerning the equations of motion or some other governing differential equations. It is of interest to utilize the basic format of the equations and perform an analysis of the available data base which will provide a best estimate of each of the unknown parameters, thus, completing the identification as required. For example, suppose that one is given a linear matrix differential equation of the form

$$\dot{x} = Ax + Bu$$

where x is a vector of n state variables, u is a vector of m control variables, A is a time invariant n x m matrix, and B is a time invariant n x m matrix. Find, from a given time history of x(t) and u(t)(and, possibly, $\dot{x}(t)$), the unknown parameters of the A and B matrix.

FINITE-STATE MACHINE NO. 1

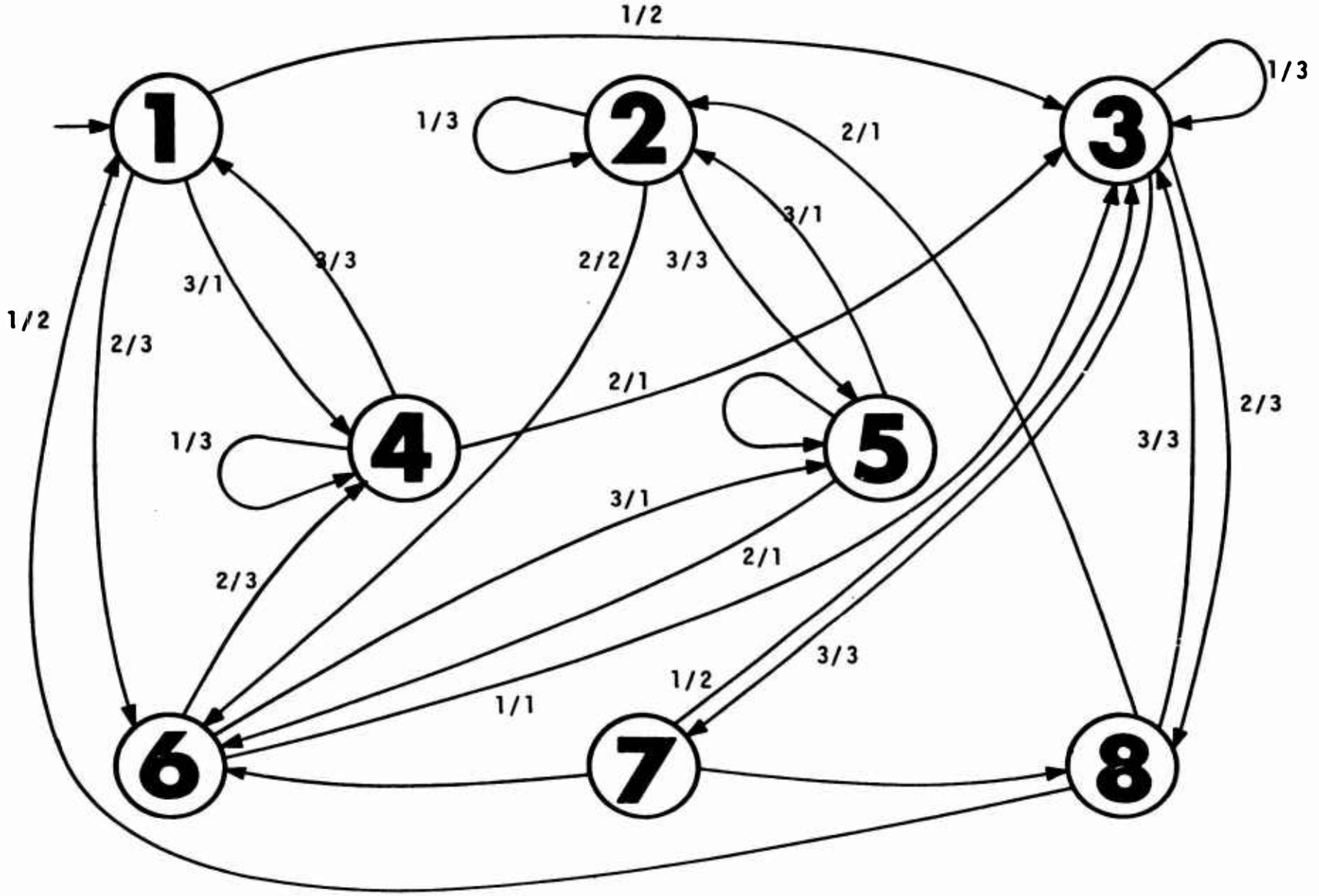


Figure 1.

FINITE-STATE MACHINE NO. 2

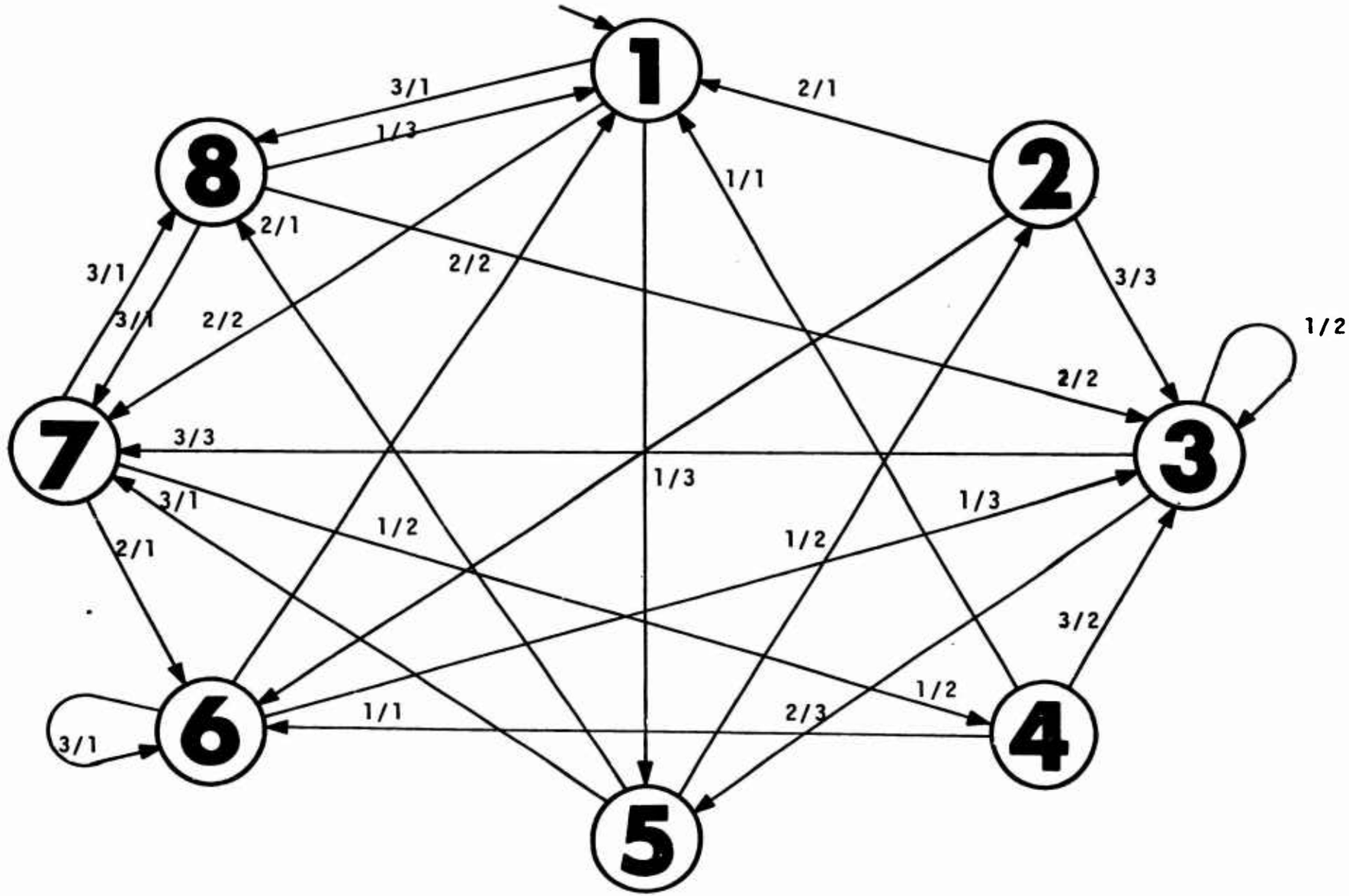


Figure 2.

Table 1.
INPUT AND OUTPUT OF MACHINE I

Input Sequence

1	1	2	3	1	3	2	1	1	3	1	2	2	2	1	3	1	3	1	3	3	2
2	1	1	3	3	3	2	3	3	3	3	1	2	1	1	2	3	1	2	3	3	2
2	2	3	1	3	3	2	2	3	2	3	1	2	2	2	2	3	2	1	1	2	1
3	1	2	2	3	3	3	3	2	2	1	3	2	1	3	3	3	2	3	1	3	2
1	3	1	1	2	2	1	1	2	2	3	3	2	3	1	3	1	2	3	3	3	1
3	3	1	1	3	3	1	3	3	2	1	2	3	2	2	3	1	2	1	2	1	1
1	1	2	1	3	3	3	1	2	1	1	1	1	1	3	2	2	3	2	2	1	3
1	3	2	2	1	1	1	3	2	1	2	3	2	2	3	2	3	1	2	3	3	1
2	1	2	2	1	1	1	1	3	2	3	1	2	1	2	2	1	3	1	2	1	2
1	3	3	1	2	3	3	1	3	2	1	2	3	2	1	1	3	2	2	3	1	2
1	1	1	2	3	1	1	2	2	1	2	1	1	1	2	3	3	2	2	3	2	2
1	1	2	2	2	3	1	2	3	2	1	2	2	3	3	2	3	3	2	1	2	3
1	1	2	2	1	2	1	2	2	1	1	3	3	2	2	2	2	1	3	3	1	1
2	3	2	1	1	3	1	1	1	3	2	2	1	3								

Output Sequence

2	3	3	3	3	3	3	1	3	3	2	3	1	2	1	3	2	3	2	3	2	1
2	1	3	3	2	3	3	3	3	2	3	3	3	2	2	3	3	3	3	3	3	3
3	1	3	2	3	2	1	2	1	1	1	2	1	3	1	3	3	3	2	2	3	2
1	3	1	3	3	3	2	3	3	1	3	3	1	1	3	2	3	3	3	3	3	3
1	3	2	3	3	1	3	3	2	3	3	1	1	3	2	3	2	3	3	3	2	2
1	3	2	3	3	2	2	1	3	3	1	3	3	3	1	3	2	1	1	3	2	2
3	3	3	2	1	3	1	3	1	3	3	3	3	3	3	3	3	3	3	3	3	3
2	3	3	3	3	3	3	3	3	3	1	3	3	3	1	3	1	1	2	1	1	1	3
2	1	3	1	3	3	3	3	3	1	1	2	1	1	3	1	3	3	2	1	1	3
2	1	3	2	3	3	3	2	1	1	3	3	3	3	2	2	3	3	3	3	2	3
2	2	3	3	3	3	3	3	1	3	2	1	3	3	3	3	3	3	3	3	3	3
3	3	1	3	1	3	2	1	1	1	1	3	1	3	1	2	1	1	2	1	3	3
3	3	3	1	3	2	1	3	1	3	3	3	1	2	3	1	3	2	1	3	2	3
3	3	3	2	2	3	2	3	3	3	3	3	3	3								

Table 2.
INPUT AND OUTPUT OF MACHINE II

Input Sequence

1	1	2	3	1	3	2	1	1	3	1	2	2	2	1	3	1	3	1	3	3	2
2	1	1	3	3	3	2	3	3	3	3	1	2	1	1	2	3	1	2	3	3	2
2	2	3	1	3	3	2	2	3	2	3	1	2	2	2	2	3	2	1	1	2	1
3	1	2	2	3	3	3	3	2	2	1	3	2	1	3	3	3	2	3	1	3	2
1	3	1	1	2	2	1	1	2	2	3	3	2	3	1	3	1	2	3	3	3	1
3	3	1	1	3	3	1	3	3	2	1	2	3	2	2	3	1	2	1	2	1	1
1	1	2	1	3	3	3	1	2	1	1	1	1	1	3	2	2	3	2	2	1	3
1	3	2	2	1	1	1	3	2	1	2	3	2	2	3	2	3	1	2	3	3	1
2	1	2	2	1	1	1	1	3	2	3	1	2	1	2	2	1	3	1	2	1	2
1	3	3	1	2	3	2	1	3	2	1	2	3	2	1	1	3	2	2	3	1	2
1	1	1	2	3	1	1	2	2	1	2	1	1	1	2	3	3	2	2	3	2	2
1	1	2	2	2	3	1	2	3	2	1	2	2	3	3	2	3	3	2	1	2	3
1	1	2	2	1	2	1	2	2	1	1	3	3	2	2	2	2	1	3	3	1	1
2	3	3	1	1	3	1	1	1	3	2	2	1	3								

Output Sequence

3	2	1	1	3	1	2	2	2	3	2	1	2	1	3	3	2	2	2	3	1	2
3	2	2	1	1	1	2	1	1	1	1	2	1	3	2	1	1	3	2	1	1	1
2	2	1	3	1	1	1	2	1	2	3	2	1	2	1	2	1	2	2	2	3	2
3	2	3	1	1	1	1	1	2	3	2	3	3	2	3	3	1	2	3	2	2	3
2	3	2	2	3	1	3	3	1	2	3	1	2	3	2	2	2	3	1	1	1	2
2	3	2	1	1	1	3	3	1	2	2	3	1	1	2	1	3	2	2	1	3	2
2	3	3	2	3	3	1	3	2	2	1	3	2	2	3	1	2	1	2	3	2	3
2	3	1	2	3	2	2	1	2	3	1	1	1	2	1	2	3	2	1	1	1	2
1	3	1	2	2	2	2	2	3	1	1	3	3	2	1	2	2	2	2	3	2	1
3	1	1	3	2	1	2	2	3	1	3	3	1	1	3	2	3	1	2	1	3	2
2	1	3	3	1	2	1	2	2	2	1	3	2	2	2	1	1	1	2	1	2	3
2	2	2	2	1	1	3	3	1	1	3	3	1	1	1	2	3	1	2	2	3	1
2	1	2	2	2	1	3	1	2	2	2	3	1	2	3	1	2	2	3	1	3	3
1	1	1	3	2	3	2	1	3	3	1	2	3	1								

	Finite-State Machine I	Finite-State Machine II
Evolutionary Program	99.3%	99.3%
Human Operator 1		63 %
Human Operator 2		41 %
Human Operator 3	40 %	

Table 3.

If a time history of $x(t)$ is generated by an exact solution of the differential equation above, a formulation of $n(n + m)$ linear equations using values of $x(t)$, $\dot{x}(t)$, and $u(t)$ at points which will guarantee linear independence between these equations permits determination of the exact values of the $n(n + m)$ unknowns of A and B.

If exact values of $\dot{x}(t)$ are not given yet $x(t)$ and $u(t)$ for Δt sufficiently small are known, numerical differentiation of $x(t)$ yields sufficiently small are known, numerical differentiation of $x(t)$ yields sufficiently accurate estimates of $\dot{x}(t)$ so that again, by formulating $n(n + m)$ linear independent equations, acceptable approximate values for the unknowns A and B can be found.

If, however, the time history of $x(t)$ is determined by a process governed by the differential equation noted above, but the sequence of values of the $x(t)$ is obtained by measurements, contaminated by noise, a solution of $n(n + m)$ independent linear equations will give only estimates of the unknowns A and B. Improved estimates can be obtained by solving, in the least square sense (or minimax, which is computationally more difficult), a system with more than $n(n + m)$ equations. This "least square" solution is also applicable and useful in the previously mentioned case where the $\dot{x}(t)$ have been obtained by numerical differentiation of $x(t)$.

These cases assumed that the values of x(t) were obtained by a process which can be exaccly described by a linear differential equation of the form described above. However, this equation may be only a first approximation to a process which is, physically, a nonlinear process. This is the case, for instance, in the equations of rotational motion of an aircraft, where we have products of p, q, and r appearing in the equations for $\dot{p}$, $\dot{q}$, and $\dot{r}$. The problem, thus, consists in finding two time invariant matrices, A and B, which allow the approximation of the nonlinear differential equations of motion by a system of linear differential equations valid over a limited period of time and range of state and control variables. The problem of determining stability derivatives from flight data falls into this last class.

Looking at the problem of system parameter identification from the point of view of function optimization, the problem can be formulated as follows: Define a payoff function. For instance,

$$z = \sum_{i=1}^{n} \int_{t_o}^{t_{end}} (x_i - \tilde{x}_i)^2 / w_i \, dt$$

Since x(t) is a function of the undetermined parameters $\alpha_1 \ldots \alpha_n (n + m)$, z is also a function of these parameters. Consider them as independent variables defining an n(n + m)

dimensional space. Here is the classical problem of minimizing a function of n(n + m) independent variables with no constraints.

This problem can be attacked in several ways: A generalized Newton-Raphson method for finding zeros of functions of several variables may be used. Several gradient methods based on steepest descent have been developed in the past. If the number of independent variables is relatively large, methods based on random search techniques have been used relatively successfully. The problem of function minimization with many independent variables is particularly difficult to solve economically when a comparatively large amount of computer time is required to evaluate z and is larger than the time required to make a decision about the necessary changes in the parameters. Here, the evolutionary program is used to find a logic which should help to find a more rapidly converging path in the optimization procedure. The program used permits adjusting two free parameters, these starting with a random search procedure and as soon as enough trials have been made using the finite-state machine evolved by the evolutionary program to decide which next move will yield an improvement in the value of the payoff function.

Situations often arise in which it is desired to have a given system or plant perform in a particular manner. The problem then is how to manipulate the input parameters so as

to achieve the desired performance. The degree of difficulty in accomplishing this result depends on many factors, for example, the complexity of the plant dynamics, how well these dynamics are known or the controllability of the input parameter. Consider the case where the input parameters are controllable but the plant dynamics are only partially known. A first step in controlling such a plant or device would be to obtain a workable model of it. Suppose that it is possible to sample the inputs and outputs at fixed time intervals so that time histories of the input-output pairs are available. For the application of the evolutionary program a natural question is whether or not the time history can be adequately modeled by a finite-stated machine. Can one find a class of plants or devices so that sampled time histories can be adequately modeled by finite-state machines? For such a class the application of evolutionary programming then becomes feasible.

Now a large class of input-output devices have their dynamics representable by differential equations. Consider those where the output $y(t)$ is related to the input $x(t)$ by a differential equation of the form $y' = F(x, y)$. Here F is assumed to be only partially known. In practice, even if F were known, closed-form solutions do not exist or are very difficult to derive so that solutions are usually obtained by numerical techniques. Such solutions consider sampled values at cer-

tain time intervals (usually fixed although sometimes variable time steps are used). In general, the solution formulas give the output value y_n at the n-th step as a function of the current input value as well as a certain number of the previous input and output values, for example,

$$y_n = f(x_n, x_{n-1}, \cdots, x_{n-k}, y_{n-1}, \cdots, y_{n-k}).$$

If the range of input and output values are quantized into a finite number of intervals, then the solution formula gives a function from a finite input alphabet to a finite output alphabet which is completely determined by the current input together with the last k input and output. As will be shown later in this report, such a function can be represented by a finite-syate machine and so finite-state machines provide approximate models for this class of devices. How well they can approximate such a device depends, of course, on the sampling rate of the input-output values as well as the interval sizes used in quantizing the range of input and output values.

In order to demonstrate the feasibility of finite-state machine controlled systems and to point out some of the difficulties which will have to be expected in such systems, a simple second-order sampled system, as shown in Figure 3, is investigated.

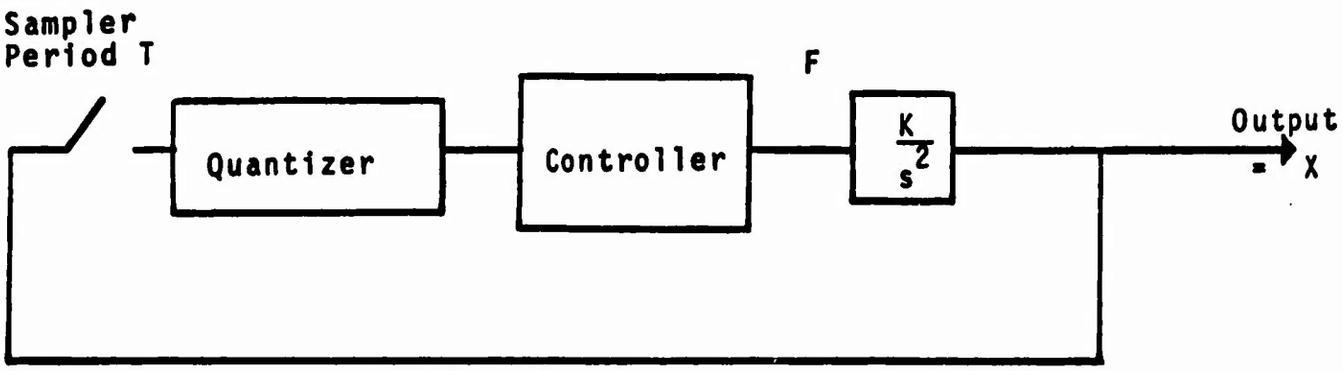


Figure 3. Second Order Sampled System

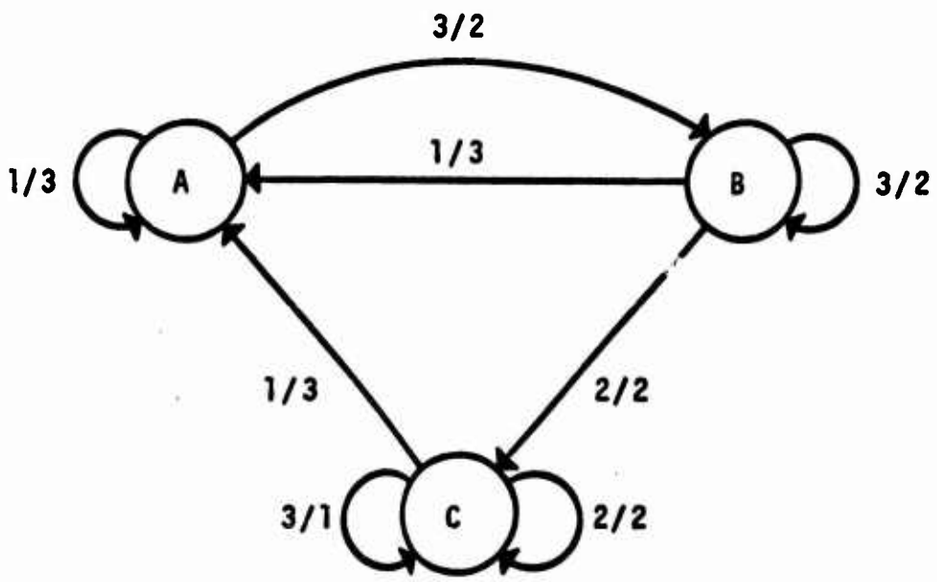


Figure 4. Finite-State Machine Controller
for Second Order System

Assume the system which has a finite-state machine as controller to work as follows: After every T seconds, the sampler is closed for a time $\tau << T$. The present value of the output variable is quantized, and for our simple example, a quantization into three levels will be assumed. The quantized state variable x determines the input symbol to the finite-state machine according to Table 4. Let the outputs of the finite-state machine be the symbols 1, 2, or 3 and let the following inputs to the second-order system correspond to the three input symbols shown in Table 5. The finite-state machine shown in Figure 4 will be a suitable controller for such a system. In the form of a state transition table, the finite-state machine is represented in Table 6. The dots in Table 6 indicate "don't care" situations. If properly synchronized, an input of 2 when the machine is in state 2 can never occur. This shows that when designing a controller in the form of a finite-state machine some thought must be given on how to properly synchronize the system. This may become one of the major design problems when the observed values of the state variables which have to be controlled are contaminated by noise. Noisy inputs might well produce transitions in the finite-state machine casuing loss of synchronization. Sometimes it is possible to design self-synchronizing finite-state machines so that even a temporary transition into

X	Input Symbol
$X \leq 0$	1
$X \geq L$	2
$0 < X < L$	3

Table 4.

Output	Force
1	0
2	-F
3	+F

Table 5.

State	Input		
	1	2	3
A	A/3	•/•	B/2
B	A/3	C/2	B/2
C	A/3	C/2	C/1

Table 6.

an improper state will eventually correct itself. Such a self-synchronizing finite-state machine may be more complex than a finite-state machine capable only of controlling a system with noise-free measured state variables. In any further analysis, some effort should concentrate on solving the control problem for noisy state variables.

Figure 5 shows the phase plane plot for a system using the above-described finite-state machine starting the machine in the properly synchronized state. The phase plane plot indicates that the system can be controlled to achieve a stable limit cycle. Note that the system, if sampled in any way indicated in Figure 3, would be unstable in general when using only one switching line, because of the time-delay between x changing its sign and the switching of the forcing function. Such a system would show a behavior as given in Figure 6.

Even a system with two switching lines where a zero force would be applied within the region $0 \leq x \leq L$ would be unstable due to the time delays of the switching; such a system would show a behavior as displayed in Figure 7. Looking back at the system controlled by the finite-state machine, we realize that the capability of the finite-state machine to recognize the path by which the system entered the state where $0 \leq x \leq L$ allows to apply zero force when passing through that state in the direction of decreasing x, but applying negative force when passing through those regions with increasing x. It be-

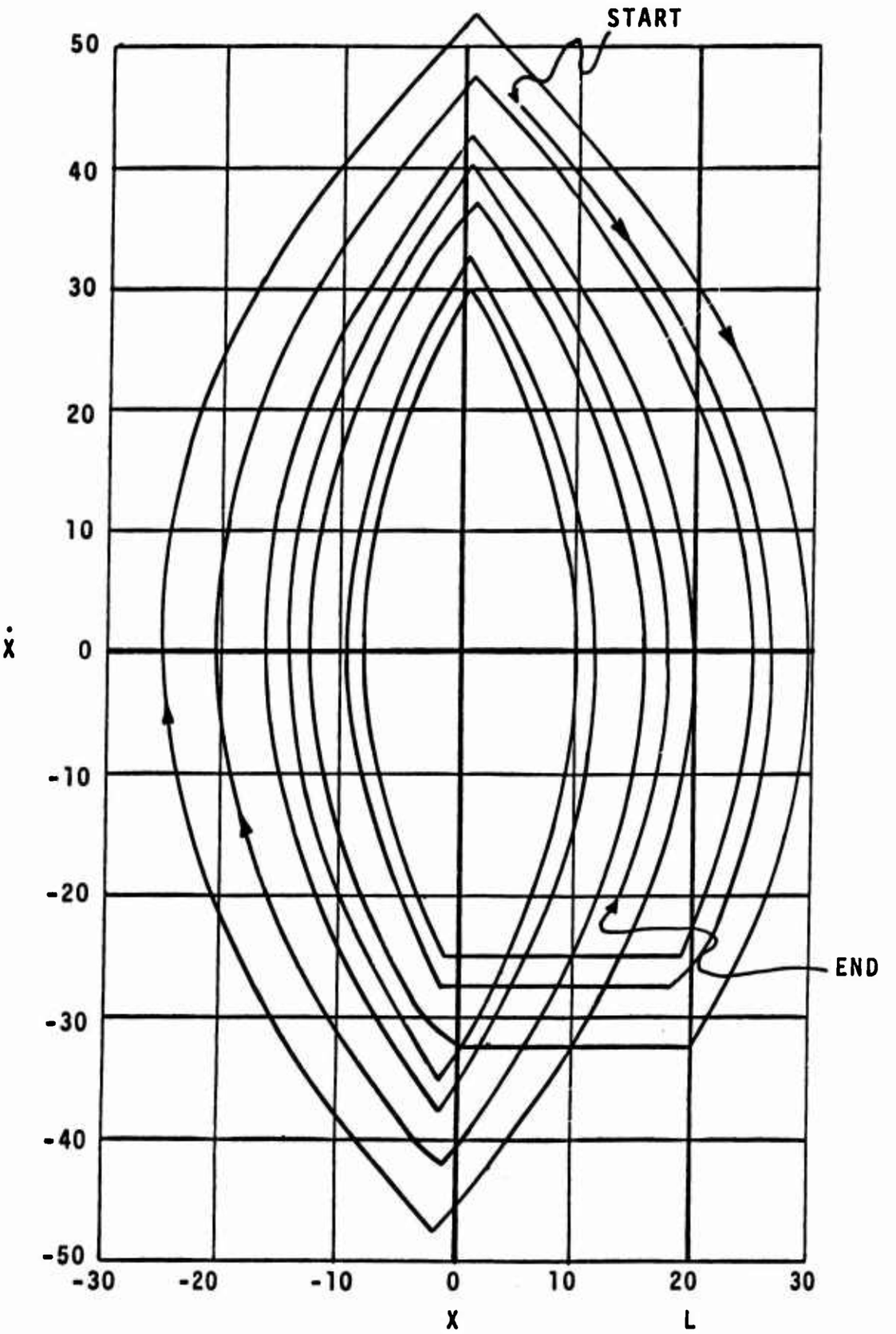


FIGURE 5.

Phase Plane Plot of Second Order Sampled System
Controlled by Finite-State Machine
K=1; T=0.1; F=50; L=20

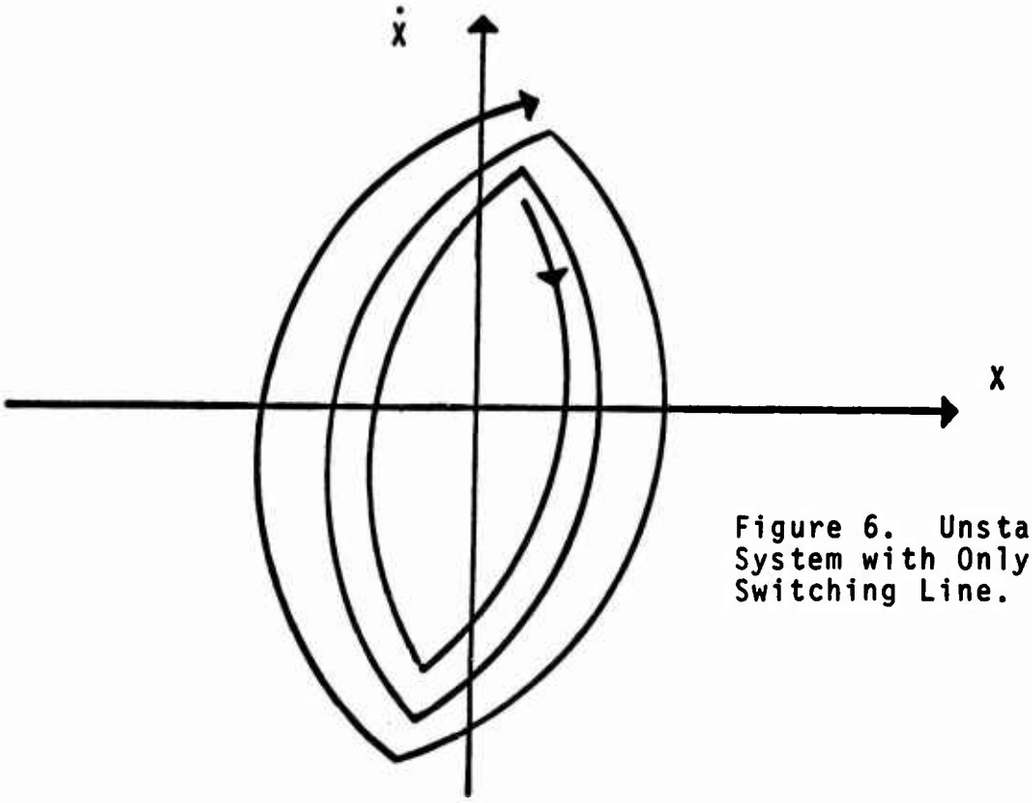


Figure 6. Unstable System with Only One Switching Line.

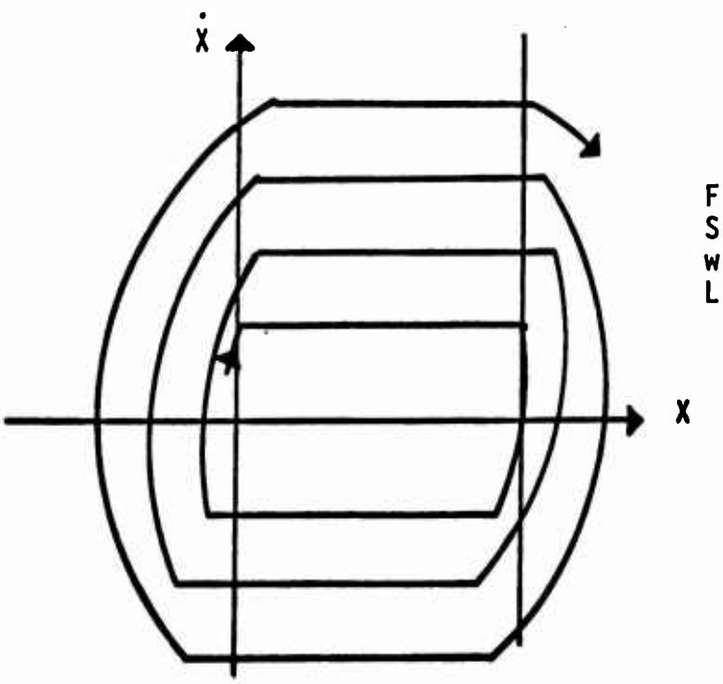


Figure 7. Unstable Second-Order System with Two Switching Lines.

comes also clear that a controller performing the same function could be designed if both x and $\dot{x}$ were available as inputs to the controller. The solution with the finite-state machine, however, has the advantage of not requiring knowledge of $\dot{x}$.

Theoretical consideration shows that within the accuracy of the quantizing of the data certain finite-state machines could serve as models of transducers whose transfer functions can be described by differential or difference equations. An example of this is the modeling by finite-state machines of a servo drive driven by a random sequence where the input and output as functions of time are known. A section of the input and output is shown in Figure 8.

Since the input is on-off, a two-level coding suffices. The output was quantized into 80 levels. Two approaches were considered for the finite-state machine. One, the input, would be binary and the past output would be reflected in the states. This would require a large number of states. The alternative would be to code the actual servo input and the past output into the finite-state machine input, thus, requiring fewer states. This latter approach was selected with an input set of 160 symbols and the output in 80 quantized levels.

Two machine models were evolved (Figure 9). A two-state machine and a four-state machine resulted. The two-state machine modeled well in the middle part of the range where the servo output was fairly linear but its accuracy fell off at

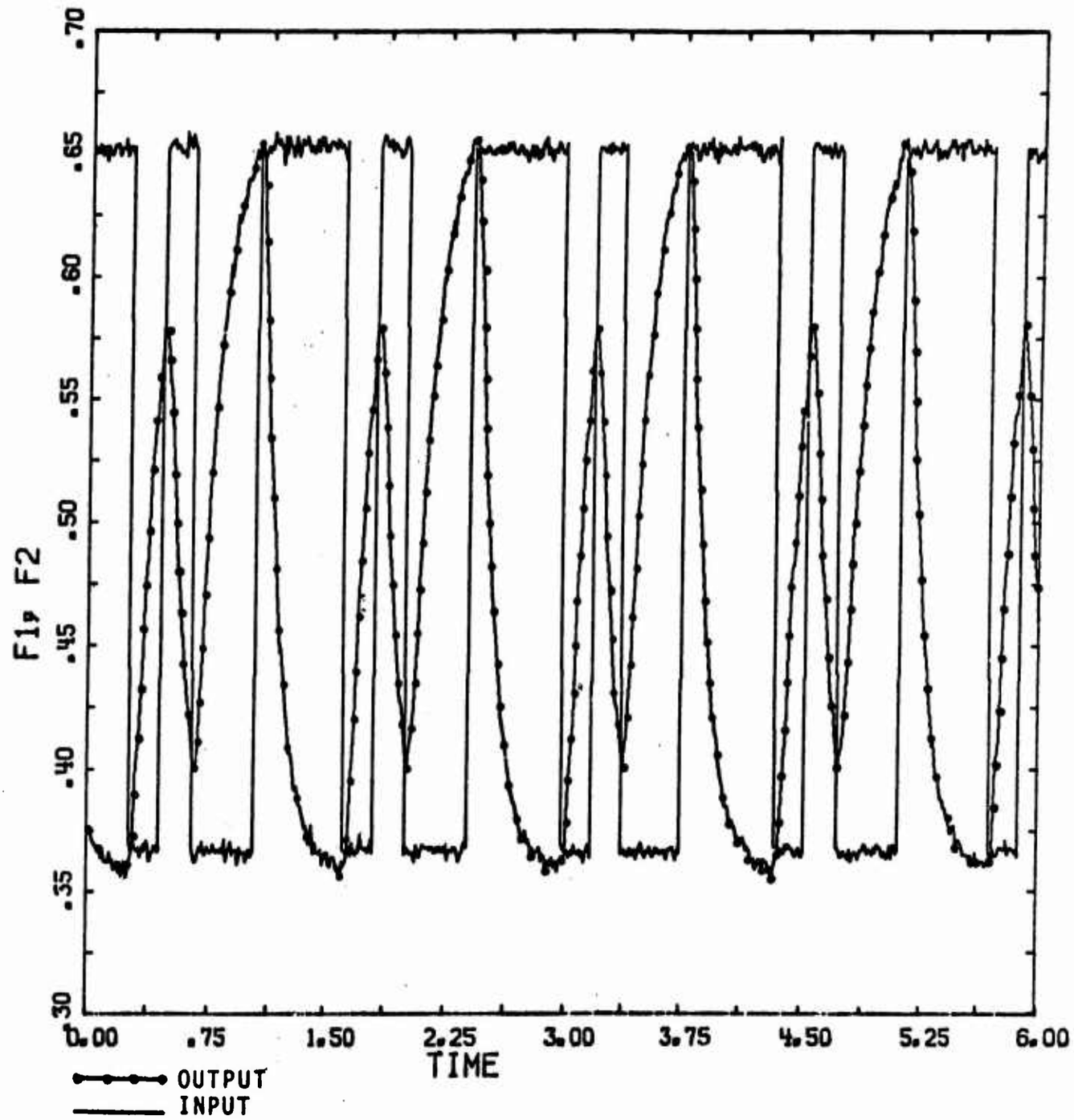
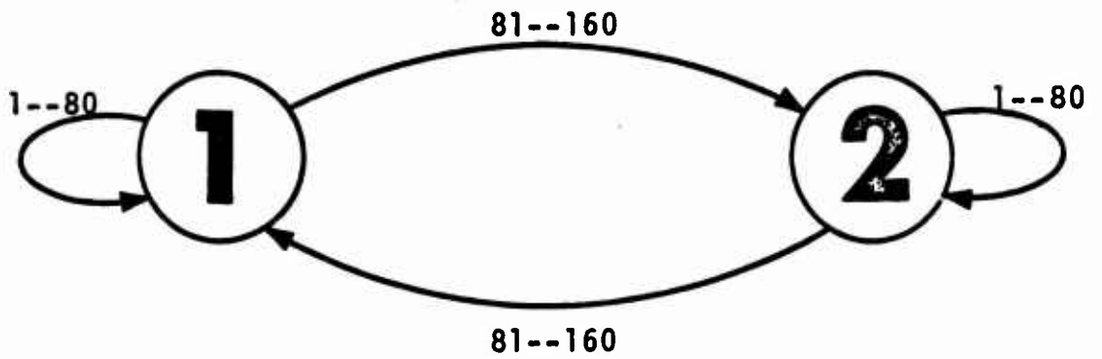
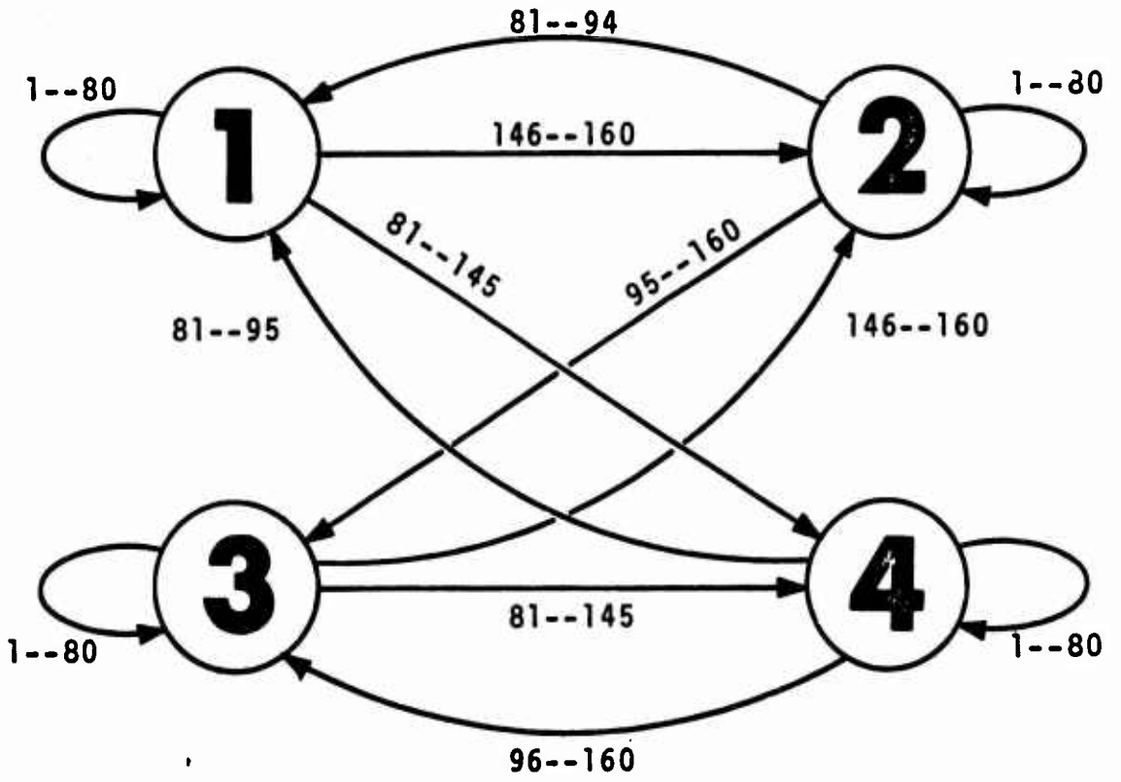

F1, F2
.70
.65
.60
.55
.50
.45
.40
.35
.30
0.00
.75
1.50
2.25
3.00
3.75
4.50
5.25
6.00
TIME
OUTPUT
INPUT

Figure 8.



(a) Two-State Machine



(b) Four-State Machine

Figure 9.

the extremes where the servo output was non-linear. The four-state machine performed as well on the linear part and better on the non-linear portion. Both models tracked the servo output over the time history with an average error of less than five percent.

The experiment also indicated some possible difficulties which must be considered in application of finite-state machines as models. One such is that the quantization levels and sampling times are too short or the quantization levels too gross, the change in output per sample time may not be reflected in the quantized output and the machine becomes locked in an output symbol. This problem occurred on the servo models at the extremes where the slope of the output curve was small. A possible remedy in the current model would be the non-linear quantization of the servo output.

The possible use of automata for controlling unknown or partially known plant systems* as shown in Figure 10 were considered. In these systems, M_1 is a fixed transducer with input I being a step input and M_2 is a transducer with transfer function of form K/f(s). K represents a gain and is a controllable input, the task being to maintain the output O(t) between two given curves, $g_1(t)$ and $g_2(t)$ for $0 \leq t \leq T$ with T fixed and t=0 corresponding to time of step input.

*The motivation for the example is a control system of an airplane with the input change of stick position.

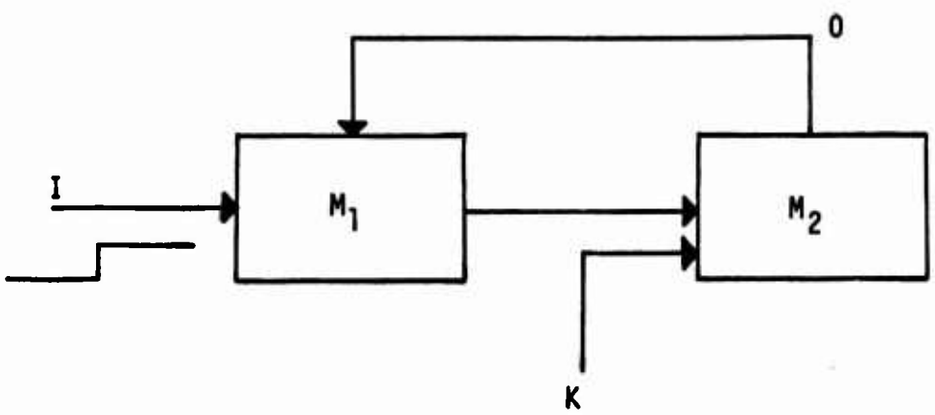


Figure 10.

If K is restricted to a finite set of values, then, as will be shown, for each value of k, the transfer function in the time domain can be represented by a finite-state machine, more exactly, by a finite input-output memory machine. A parallel composition of these machines would then give a model of M_2 permitting, at least in principle, a decision algorithm to determine a best or an allowable value for K.

Now such an approach, particularly if K can assume a large number of values, would be cumbersome. Moreover, since each machine is actuated by the same input-output sequence, few of the total number of states in the parallel composition are even exercised. But since the parallel composition of finite input-output memory machines is also of this same type, an alternative approach is to have a single such machine as a model using many different training runs to evolve it.

It is important to consider extended memory machines and embedded machines. An extended memory machine is a machine, the input of which consists of the input symbol sequence together with some combination of the last few input symbols and the last few outputs of the machine. For reference purposes, denote the combination of previous input and output symbols as the extended memory.

For such a machine with a given start state and a given initial extended memory, there is an equivalent finite-state

machine without extended memory which has been called an embedded machine. In general, different initial states and different initial extended memory can give rise to different embedded machines.

The original technique for exhibiting the existence of the embedded machines involved the finite-state functions associated with these machines. While theoretically convenient for proving existence of the embedded machine, this technique was difficult to implement and apply to the problem of extracting the embedded machine. Since in applications to modeling, extended memory machines were often used, a simpler scheme for extracting the embedded machine was desired. Efforts were made to devise one, this resulting in the technique described below.

For simplicity the technique will first be described for a machine whose extended memory consists of the last output symbol. Let q_i be a state of the extended memory machine and let o_j be a symbol which is an output for some transition into q_i. Then the pair $q_i o_j$ gives rise to a possible state for the embedded machines as does any other such state-output pair. For such a state the output and next state transition associated with each input symbol will now be given.

Let a be any input symbol; then for the state $q_i o_j$ the output is that output when state q_i is given the input $(o_j a)$. Suppose it is o_k. Let q_n be the next state reference when

$(o_j a)$ is the input to state q_i. Then the next state reference for state $q_i o_j$ with input a is the state $q_n o_k$. Note that this satisfies the requirements for a possible state of the embedded machine. The actual embedded machine is the submachine consisting of those states which can be reached from the state corresponding to the next state and the initial extended memory.

Consider the example defined by Table 7 of the machine with extended memory consisting of the last output symbol. This is a five-state machine with start-state 3 and initial extended momory of 1. An input pair consists of the ordered pair of the last output and current input. The entries in the body of the table give next state and current output in that order. For example, if the machine is in state 2 with input of 13, it transfers to state 4 and outputs a 3. Table 8 lists the states which might be states of the embedded machine. Entries in the table are interpreted as follows: The pair listed in the next state, the second element in the pair, is the output. A check of the table shows that states 11, 12, and 31 are indistinguishable states and can be combined into one state which is called 31 (since this is the start state). Also, states 41, 52, and 53 can never be reached from the start state and so can be dropped.

The resulting machine, which is the embedded machine, is given by Table 9 with (a) being the next state reference portion and (k) the output portion. Although the machine of

TABLE 7.

	11	12	13	21	22	23	31	32	33
1	11	23	31	11	23	11	43	21	22
2	22	32	43	22	12	42	12	42	23
3	31	23	11	12	33	21	11	22	51
4	11	33	41	13	42	42	22	33	31
5	22	32	23	13	11	42	33	42	23

Start State - 3

Initial External Memory - 1

TABLE 8.

	1	2	3
11	11	23	31
12	11	23	11
13	43	21	22
21	22	32	43
22	22	12	42
23	12	42	23
31	31	23	11
32	12	33	21
33	11	22	51
41	11	33	41
42	13	42	42
43	22	33	31
51	22	32	23
52	13	11	42
53	33	42	23

START STATE 31

TABLE 9.

	1	2	3		1	2	3
13	43	21	22		3	1	2
21	22	32	43		2	2	3
22	22	31	42		2	2	2
23	31	42	23		2	2	3
31	31	23	31		1	3	1
32	31	33	21		2	3	1
33	31	22	51		1	2	1
42	13	42	42		3	2	2
43	22	33	31		2	3	1
51	22	32	23		2	2	3

(a) (b)

START STATE 31

Table 5 is a state output machine, the combining of states 11, 12, and 31 causes the resulting machine not to be a state output machine. Hence, the output table is also required.

Consider now the case of the general extended memory machine with extended memory of, say, the last k output symbols and the last j input symbols. A typical input would then be

$$o_1 \ldots o_k a_1 \ldots a_j a$$

with the subscripted portion being the extended memory. A possible state in the embedded machine would be denoted by the sequence

$$q_i o_1 \ldots o_k a_1 \ldots a_j$$

where q_i is a state of the extended memory machine and

$$o_1 \ldots o_k a_1 \ldots a_j$$

is a sequence of inputs and outputs whose last transition is into q_i. The output for this state with input a would be the output of state q_i with

$$o_1 \ldots o_k \ldots a_j a.$$

Suppose it is o. Further, let q_n be the next state reference when q_1 has input

$$o_1 \ldots o_k a_1 \ldots a_j.$$

Then the next state reference for the state

$$q_i o_1 \ldots o_k a_1 \ldots a_j$$

with input a is

$$q_n o_2 \ldots o_k o a_2 \ldots a_j a.$$

It is well known that difference equations can be approximately represented by differential equations. These in turn can be represented by finite-state machines within some quantization error. In a differential equation the n-th output is represented as a function of some finite number of previous inputs and outputs. It will now be shown that such a function can be represented by a finite-state machine.

Formally, if A and B are finite sets and ΣA is the class of finite sequences of elements in A, then f is such a function if f maps ΣA into B and if there is a function g on $A^{k+1} \times B^k$ into B so that for $n > k$,

$$f(a_n, \ldots, a_1) = g(a_n, a_{n-1}, \ldots, a_{n-k}, b_{n-1}, \ldots, b_{n-k})$$

where

$$b_i = f(a_i, \ldots, a_1) \quad \text{for} \quad i = 1, \ldots, n-1.$$

(A^{k+1} is the set of sequences of length k+1 for A and B^k is the set of sequences of length k from B). That such a function f can be realized by a finite-state machine will be shown by constructing a machine which will do so.

Consider the machine $m = (A, B, S, s_0, \lambda, \delta)$ where S consists of the initial state s_0, a state for each sequence

$$(a_j, \ldots, a_i, b_j, \ldots, b_1) \quad \text{where} \quad 1 \leq j < k \quad \text{and}$$

$$b_i = f(a_i, \ldots, a_1) \quad \text{for} \quad 1 \leq i \leq j$$

and a state for each sequence

$$(a_k, \ldots, a_1, b_k, \ldots, b_1).$$

Not all k-tuples of b's are necessary, only those which are possible values of f for the corresponding a's, but the extraneous states pose no problem. The next state function λ

is defined by

$$\lambda(a, s_0) = (a, b) \quad \text{where} \quad b = f(a)$$

$$\lambda(a,(a_j,\ldots,a_1, b_j,\ldots,b_1)) = (a,a_j,\ldots,a_1, b, b_j,\ldots,b_1)$$

for $1 \leq j < k$ where $b = f(a, a_j, \ldots, a_1)$.

$$\lambda(a, (a_k, \ldots, a_1, b_k, \ldots, b_1)) = (a, a_k, \ldots, a_2,$$

$$b, \ldots, b_2)$$

where

$$b = g(a, a_k, \ldots, a_1, b_k, \ldots, b_1).$$

The output function δ is defined by

$$\delta(a, s_0) = f(a)$$

$$\delta(a, (a_j, \ldots, a_1, b_j, \ldots, b_1)) = f(a, a_j, \ldots, a_1)$$

for $1 \leq k < k$

$$\delta(a,(a_k,\ldots a_1,b_k,\ldots b_1)) = g(a,a_k,\ldots,a_1,b_k,\ldots,b_1).$$

For any sequence of α from ΣA, a test by cases on the length of α shows $\hat{\delta}(\alpha, s_0) = f(\alpha)$ where $\hat{\delta}$ is the extension of δ from A to ΣA, that is, the machine M realizes the function f. In general, the machine M is not reduced so that an equivalent machine with fewer states would also realize the function. Machines which realize such functions are termed finite input output machines with memory length k. The above-constructed machine has not more than

$$1 + (pq) + (pq)^2 + \ldots + (pq)^k$$

states where p is the number of input symbols and q is the number of output symbols. On the other hand, Massey[1] has exhibited for any n, machines with n states and input-output memory of length n(n-1)/2. Gill[2] has shown that for any n-state machine, this is the maximum memory length.

It can also be shown that the series or parallel composition of such finite input-output memory machines are finite input-output machines. Since the evolutionary program mutates

[1]Massey, James L. IEEE Transactions on Electronic Computers, Vol. EC-15, No. 4, August 1966, pages 658-659.

[2]Gill, A. Introduction to Theory of Finite-State Machines, New York, McGraw-Hill, Inc., 1962.

the underlying state structure or automata, if the finite input-output machines have a well-defined underlying class of automata, restricting the mutation to this subclass should improve the efficiency of the evolutionary program in these cases where a finite input-output machine is the desired one. This was investigated. A class of automata was found which for any output function yields finite input-output machines; however, they are restricted in that the input alone determines the output, for example, for any input sequence of length k only one output sequence is associated with it. Such machines would not suffice, for example, to model devices whose input-output function satisfies a differential equation.

On the other hand, for any automata if the output alphabet equals or exceeds the number of states, there exists output functions which yield finite input-output machines. To see this, let δ be any input function such that for each input symbol a and each pair of distinct states, s_1 and s_2, $\delta(a, s_1) \neq \delta(a, s_2)$. Since there are as many output symbols as states, this is possible. Hence, for any given input symbol, the resulting output symbol uniquely determines the state and the machine has input-output memory of length two. It, therefore, appears unlikely that a suitable subclass exists.

Finite-state machines were used in all applications of the evolutionary program. More powerful logical entities such as Turing machines were considered, however, inherent difficulties such as the "halting" problem discouraged their use. It was, therefore, appropriate to investigate logical devices intermediate to finite-state machines and Turing machines. One candidate in this regard is pushdown automata. These automata should be more powerful than finite-state machines in that they hold some explicit memory. Moreover, they are not subject to the halting problem. In view of the fact that mutation within the evolutionary program is applied to the overlying finite-state automata (the output symbols being deterministically generated so as to provide the best fit of the data for the given automata), it appears particularly appropriate to consider pushdown automata.

A pushdown automata M is a 6-tuple, $(A, C, Q, \delta, z_0, q_0)$, where A is the input language, C is the pushdown alphabet, Q is the set of states, z_0 is a special pushdown symbol, q_0 is the initial state, and δ is a mapping of AXCXQ into C*xQ. (C* is the set of all finite words which can be made from the alphabet C including the empty word e.) Intuitively, with each triple consisting of an input symbol, a pushdown symbol, and a state, δ associates a pair consisting of a word made up of symbols from the pushdown alphabet and a state (the next state). Translating this into machine opera-

tion, suppose the automata has the word w in the pushdown store where w = xc with x a word in C* and c in C, is in state q_1 and is given the input symbol a. If $\delta(a, c, q_1) = (y, q_2)$, then the automata goes into state q_2 and the word xy is now in the pushdown store. Note that y may be the empty word in which case the word in the pushdown store would be x, for example, the symbol c is "erased." Under this convention, it is the last symbol in the pushdown store that is read so that the pushdown store is essentially a last-in, first-out device. The symbol z_0 is the initial symbol in the pushdown store and to prevent emptying of the store and the resulting premature halting of the automata, erasing of z_0 is no allowed, that is, it is never the case that $\delta(a, z_0, q) = (e, q')$.

The question remains as to whether or not pushdown automata are more powerful than finite-state machines when used as predictors. Only a partial answer to this question can be offered. Note that a finite-state machine, when used as a predictor (that is, the last output symbol is used as the next input symbol), generates output sequences which are eventually periodic. It is of interest to determine whether or not pushdown automata, when used as predictors, will generate only sequences that are eventually periodic.

The action of a pushdown store automaton is determined by a triple consisting of the present state of the automaton,

the input and the top symbol in the pushdown store. The automaton then transfers to a next state (possibly the same), outputs a symbol, and changes the pushdown store. Three different types of changes in the pushdown store can occur. The top symbol may be erased and the next symbol from the top becomes the top symbol, the top symbol may be replaced by a single symbol, or the top symbol may be replaced by a word of length greater than one. Call a triple which results in an erasure a decreasing triple and a triple which results in the top symbol being replaced by a word of more than one symbol an increasing triple.

If a pushdown store automata has no increasing triples, then only the initial pushdown symbol ever occurs in the pushdown store, and the automata is effectively a finite-state machine. So there is no loss in generality in assuming the automata has at least one increasing triple. A sequence of successive triples for an automaton is called an increasing sequence if it contains at least one increasing triple but no decreasing triples. Similarly, a sequence is a decreasing sequence if it contains at least one decreasing triple but no increasing triples.

Lemma 1. When used as a predictor, a pushdown store automata has increasing sequences of bounded length or it has an increasing sequence of infinite length. If it enters an increasing sequence of infinite length, then the output

is eventually periodic.

There are only a finite number of states, of input symbols, and of pushdown symbols. There are, therefore, only a finite number of increasing triples, say, k of them. Similarly, there are only a finite number, m, of triples which are neither increasing nor decreasing. Suppose there exists an increasing triple of length at least k + m + 1, then either an increasing triple or one which is neither increasing nor decreasing must be repeated. Since in a sequence of triples none of which is decreasing, only the top pushdown symbols are ever used, the sequence of triples between repetitions of a triple is then repeated. The resulting increasing sequence is then infinite and the output cyclic.

Lemma 2. When used as a predictor, if a pushdown store automata has decreasing sequences of triples of bounded length, the output is eventually cyclic.

If there is an infinite increasing sequence which can be entered, then Lemma 2 follows from Lemma 1. Consider then the case when all increasing sequences are bounded. Suppose the bound on the length of the decreasing sequences is k, then if the automata is actuated by a decreasing triple the ensuing actions of the automata until an increasing triple is encountered are determined by the current triple and the top k symbols in the pushdown store or by the word in the pushdown store if it contains less than k symbols. Since there are only a

finite number of decreasing triples and a finite number of k-tuples of pushdown symbols, the automata eventually must have the current triple and the top k symbols on the pushdown store a repetition of a previous one and so a cyclic output from that point on.

As yet unresolved is the case of an automata which when used as a predictor has increasing sequences of bounded length but decreasing sequences of unbounded length. If such a machine could exist then the possibility of other than eventually cyclic output exists. Efforts to devise such a machine or to prove it cannot exist have as yet been unsuccessful.

Thus, pushdown automata appear to be more powerful than finite-state machines as predictors even though the output of such a machine when used as a precictor is eventually periodic. Specifically, pushdown automata have the advantage that, with a given number of states, longer cycles can be generated than by finite-state machines.

The next logical device to be considered is the stack automata. Again, as in the case of both the finite-state machines and pushdown automata, only the deterministic form of the automata will be considered. The stact automata has the same basic structure as the pushdown automata with the additional ability to read any symbol in the stack, not just the topmost symbol. A generalization of the stack automata has also been introduced in which the input tape can be run both

forward and backward or remain stationary and is called a two-way stack automata. The form considered here is the one-way stack automata in which only forward movement or no movement is permitted. The two-way stack is prone to the "halting problem." Since the stack pointer or reader moves one symbol at a time, the stationarity of the input tape is required so as to allow the machine the potential of reading any symbol on the pushdown stack.

To show that a stack automata is actually more powerful than the pushdown automata, consider the set $\{wcw \mid$ where w is any word in $\{a, b\}^*$ and c is a symbol different from a and $b\}$. It is known[3] that this set of words is not a context-free language and, hence, is not acceptable by a pushdown automata. However, it is acceptable by the stack automata which is now described.

The stack automata consists of four states, (1, 2, 3, and 4), with 1 as the initial state and $\{3\}$ as the set of terminal states, for example, 3 is the only terminal state. In the initial state it duplicates the input symbol on the pushdown stack, moves to the next stack position and next input position, and remains in state 1 unless the symbol is a c or a

[3] Ginsburg, Seymour. The Mathematical Theory of Context-Free Languages, McGraw-Hill, Inc., New York, 1966.

blank which indicates end of input. Hence, any word w without a c in it is duplicated in the pushdown store. If a blank is input, the machine stays in state 1 and stops. When a c is the input in state 1, the machine moves the stack pointer back to the first stack symbol, moves the input ahead one, and goes to state 2. In state 2, as long as the input tape is not blank and the input tape and stack agree, it moves the stack and input ahead one position and remains in state 2. If the symbols disagree, it goes to state 4 where it stays for any input and stack symbol stopping the machine on a blank input. If the machine is still in state 2 when end of tape is reached, for example, a blank is input, and if stack symbol is blank, it moves to state 3 and machine stops, otherwise, it stays in state 2 and stops. Hence, the machine reaches state 3 and stops there only if the input is of form wcw with w in $\{a, b\}^*$.

Intuitively, the pushdown store cannot recognize a word of form wcw since, although w can be stored, the machine reads it backward and so, except for polindromes, cannot correctly compare the word after the c with the word in front. Note that a pushdown store can accept $\{wcw^R \mid \text{where } w^R \text{ is the word w reversed}\}$ since reading w^R backwards is equivalent to reading w forward.

The usual formulation for the stack automata requires that the input alphabet A contain a special symbol β, the terminal symbol. A word w from $\{A - \{\beta\}\}^*$, for example, w contains no β's, is accepted by M if whenever the word $w\beta$ is input with M

in state s_0, then when symbol β is read the machine enters one of the terminal states. The symbol β is not necessary, but simplifies the formulation. For a two-way stack automata an initial input symbol is needed to prevent running off the beginning of the tape.

When used as a transducer or predictor, the terminal symbol is unnecessary since in this application there is no concern for a last input symbol. Also, the set of terminal states is replaced by an output language C and the result of δ or $\delta_\wedge$ is now a quadruple rather than a triple. The first three coordinates are as before (for example, next state, movement of input tape, movement of stack pointer, and any writing or erasing of stack symbols) with the fourth coordinate the output symbol. One of the possible output symbols is the blank or equivalently no output so as to allow the stack pointer to move more than one space before an output is required.

Previous analysis indicated that when used as a predictor the pushdown automata would output only cyclic sequences and so does not improve on the finite-state machine as a predictor. A stack automaton will now be given which outputs as a predictor the sequence 0 1 0 0 1 1 0 ..., for example, alternate sequences of 0's and 1's of increasing length, which is certainly not cyclic.

Let M be the stack automaton with

$$S = \{s_0, s_1, s_2, s_3\}$$

$$A = \{0, 1\}$$

$$B = \{0, 1, z_0\}$$

$$C = \{o, 1, \Lambda\}$$

and δ and δ' defined as follows:

$$\delta(s_0, \cdot, z_0) = (s_0, N, R, \Lambda)^*$$

$$\delta(s_0, \cdot, 1) = (s_0, R, R, 1)$$

$$\delta'(s_0, \cdot, \Lambda) = (s_1, N, N\ 1, \Lambda)^{**}$$

$$\delta(s_1, \cdot, 1) = (s_1, N, 2, \Lambda)$$

$$\delta(s_1, \cdot, z_0) = (s_2, N, R, \Lambda)$$

$$\delta(s_2, \cdot, 1) = (s_2, R, R, 0)$$

*The · indicates either a 0 or a 1 can be entered.

**N 1 indicates no move and writing a 1 on the blank space being read.

$$\delta_{\wedge}(s_2, \cdot, \Lambda) = (s_3, N, L, \Lambda)$$

$$\delta(s_3, \cdot, 1) = (s_3, N, L, \Lambda)$$

$$\delta(s_3, \cdot, z_0) = (s_0, N, R, \Lambda)$$

The remaining triples on which δ and $\delta_{\wedge}$ should be defined cannot occur and so are not included.

A description of the machine's actions follows. At the start it is in state s_0 with the 0 or 1 as input symbol and stack symbol z_0. It stays in state s_0 with same input, but moves stack symbol ahead to next space which, at this time, is blank. It then prints a 1 in this blank space, does not move input tape or output a symbol, and moves to state s_1. Other wise, when it enters s_0, the stack pointer is on the bottom 1, the machine then moves both the input tape and stack pointers right and outputs a 1 for each 1 on the stack. When the pointer leaves the word on the stack to the first blank, the machine prints a 1 there, does not move the input tape, does not make an output, and goes to state s_1. In state s_1 there is no input tape movement nor output, the machine moves the stack pointer back until it reaches z_0 staying in state s_1. When the pointer reaches z_0, it is returned to the bottom 1 on the stack and the machine goes to state s_2 without

moving the input tape or outputing a symbol. In s_2 the machine's actions are as in s_0 except the output is a 0 rather than a 1 and when the stack pointer reaches the first blank space, the machine moves it back to the topmost 1 on the stack and goes to state s_3. State s_3 is a duplicate of state s_1 except when the stack pointer reaches the symbol z_0, the machine moves to state s_0 rather than state s_2. This completes a cycle and the machine goes through the cycle over and over except during each cycle the number of 1's on the stack is increased by one and consequently so is the length of the output sequences of zeroes and ones. For convenience, this machine was constructed so as to be independent of the input. By making the output depend on the input, different output sequences would result depending on the initial input symbol.

CONCLUSION

Finite-state machines provide a natural basis for the prediction of time series and the modeling of an unknown transducer. Evolutionary programming offers a means for finding suitable finite-state machines with respect to a given goal (payoff or error-cost matrix) so long as the cost of required data processing does not become excessive. Tracing the average cost of computation in the recent past makes it reasonable to expect much greater efficiency in coming years. The advent of intrinsic parallel computers would open new prospects for the realization of prediction and modeling through the continual search for automata which are most appropriate to the situation at hand. In fact, such mechanisms might permit improving population property constraints on the inheritance of the evolving organisms.

Various possibilities for governing the nature and amount of mutation "noise" have been considered. However, here there is an essential trade-off between efficiency and security...the more tightly linked the mutation noise to the successful kinds of mutations which have occured in the past, the less the likelihood of finding radical departures in terms of the logic of prediction and control. The greater the efficiency in the search for each new logic within a restricted class of poten-

tial environments, the less the versatility of the program and the more predictable becomes its behavior. In the face of a competitive environment, it may well prove worthwhile to be less efficient and most versatile, even at far greater computation cost.

Evolution, indeed, the scientific method, occurs at various levels of abstraction. It is clearly worthwhile to restrict attention to those models which have, in the past, been found most worthwhile and to model these as a basis for further prediction and control. In fact, models of models can be used at all levels, provided the sample size is adequate. Difficulty is encountered, however, in that the alphabet of possible symbols is much greater at each higher level. Wise judgment as to the use of a reduced alphabet of descriptors may prove worthwhile or the very essence of the higher-level modeling may have been lost. Such selection of an appropriate alphabet reduction forms a meta-problem which can only be dealt with within a particular well-defined frame of reference. Such evolutionary programming remains to be explored.

The immediate problem of this investigation has been to devise new means for solving immediate real world problems of Naval interest. Thus, with this intent, attention has been focused upon developing an improved logical capability with respect to a wide class of practical problems. Hopefully, this

technique can also be used to investigate the sensitivity of prediction and control with respect to various goals. Hopefully, it may require explicit consideration of the problem of choosing meaningful descriptors and goal formulation.

REFERENCES

1. Artificial Intelligence Through Simulated Evolution, L. J. Fogel, A. J. Owens, and M. J. Walsh, John Wiley & Sons, Inc., New York, 1966.

1a. Artificial Intelligence and Evolutionary Modeling, in Russian, a translation of Artificial Intelligence Through Simulated Evolution, by L. J. Fogel, A. J. Owens, and M. J. Walsh, John Wiley & Sons, 1966, Peace Publishers, Moscow, 1969.

2. "Artificial Intelligence Through a Simulation of Evolution," by L. J. Fogel, A. J. Owens, and M. J. Walsh, Chapter 14 of Biophysics and Cybernetic Sciences Symposium, edited by M. Maxfield, A. Callahan, and L. J. Fogel, Spartan Book Company, Washington, D. C., pages 131-155.

3. "Intelligent Decision-Making Through a Simulation of Evolution," by L. J. Fogel, A. J. Owens, and M. J. Walsh, IEEE Transactions of the Professional Technical Group on Human Factors in Electronics, Vol. 6, No. 1, pages 13-23; reprinted in Simulation, Vol. 5, No. 4, pages 267-279; and in Behavioral Science, Vol. 11, No. 4, July 1966, pages 253-272.

4. Adaption of Evolutionary Programming to the Prediction of Solar Flares, NASA Report No. CR-417, by L. J. Fogel, A. J. Owens, and M. J. Walsh, Clearinghouse for Federal Scientific and Technical Information, Springfield, Virginia, 1966.

5. "Application of Evolutionary Programming," by L. J. Fogel, A. J. Owens, and M. J. Walsh, 1966 Record of the IEEE Systems Science and Cybernetics Conference, October 17 and 18, 1966, Washington, D. C.

6. Evolution of Finite Automata for Prediction, by L. J. Fogel, A. J. Owens, and M. J. Walsh, Technical Report No. RADC-TR-66-69, USAF, June 1966.

7. "Inanimate Intellect Through Evolution," Naval Research Reviews, Vol. XX, No. 11, November 1967, pages 9-18.

8. On the Design of Conscious Automata, Final Report on Contract No. N67-20632, 31 October 1966, 102 p. refs., (AFOSR-66-2872; AD-644204), Vol. 5, No. 10, N67-20381-N67-21880, May 23, 1967.

9. "Extending Communication and Control Through Simulated Evolution," reprinted from Bioengineering--An Engineering View, edited by G. Bugliarello, San Francisco Press, Inc., San Francisco, 1968.

10. Modeling the Human Operator with Finite-State Machines, by L. J. Fogel and R. A. Moore, Final Report on Contract No. NAS1-6739, July 1968.

11. "Toward the Design of Self-Sufficient Protheses," Chapter 11 of Biocybernetics of the Central Nervous System, edited by L. D. Proctor; Little, Brown, and Company, Boston, 1969.

12. Competitive Goal-Seeking Through Evolutionary Programming, by L. J. Fogel and G. H. Burgin, Final Report on Contract No. AF19-(628)-5927, February 1969.

UNCLASSIFIED
Security Classification

AD-714201

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
Decision Science, Inc.
4508 Mission Bay Drive
San Diego, California 92109

2a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

2b. GROUP
N/A

3. REPORT TITLE
Prediction and Control Through the Use of Automata and Their Evolution

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Final Report

5. AUTHOR(S) *(Last name, first name, initial)*
Walsh, Michael J.
Burgin, George H.
Fogel, Lawrence J.

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
September 15, 1970	61	3

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
N00014-66-C-0284	
b. PROJECT NO.	
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d.	

10. AVAILABILITY/LIMITATION NOTICES
This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
N/A	Department of the Navy Office of Naval Research Washington, D. C. 20360

13. ABSTRACT
The concept of evolutionary programming was conceived as a means to find a most appropriate finite-state machine for the purpose of prediction or modeling in terms of an available data base and a payoff or error-cost matrix. Formulation and development of this concept can be traced through various publications. In this work, the use and evolution of various types of automata in prediction and control were considered. Certain problems were recognized and resolved while others were identified and brought into closer perspective. The report also includes specific findings which serve to clarify the capability and limitations of using automata and evolutionary programming for the purpose of prediction and control.

DD FORM 1 JAN 64 1473 0101-807-6800

UNCLASSIFIED
Security Classification

Reproduced by
NATIONAL TECHNICAL
INFORMATION SERVICE
Springfield, Va. 22151

UNCLASSIFIED
Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Automata Theory Control Prediction Differential Equation						

INSTRUCTIONS

1. ORIGINATING ACTIVITY: Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization *(corporate author)* issuing the report.

2*a*. REPORT SECURITY CLASSIFICATION: Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.

2*b*. GROUP: Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.

3. REPORT TITLE: Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.

4. DESCRIPTIVE NOTES: If appropriate, enter the type of report, e.g., interim, progress, summary, annual, or final. Give the inclusive dates when a specific reporting period is covered.

5. AUTHOR(S): Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.

6. REPORT DATE: Enter the date of the report as day, month, year; or month, year. If more than one date appears on the report, use date of publication.

7*a*. TOTAL NUMBER OF PAGES: The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7*b*. NUMBER OF REFERENCES: Enter the total number of references cited in the report.

8*a*. CONTRACT OR GRANT NUMBER: If appropriate, enter the applicable number of the contract or grant under which the report was written.

8*b*, 8*c*, & 8*d*. PROJECT NUMBER: Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.

9*a*. ORIGINATOR'S REPORT NUMBER(S): Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.

9*b*. OTHER REPORT NUMBER(S): If the report has been assigned any other report numbers *(either by the originator or by the sponsor)*, also enter this number(s).

10. AVAILABILITY/LIMITATION NOTICES: Enter any limitations on further dissemination of the report, other than those imposed by security classification, using standard statements such as:

(1) "Qualified requesters may obtain copies of this report from DDC."

(2) "Foreign announcement and dissemination of this report by DDC is not authorized."

(3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through _____."

(4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through _____."

(5) "All distribution of this report is controlled. Qualified DDC users shall request through _____."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. SUPPLEMENTARY NOTES: Use for additional explanatory notes.

12. SPONSORING MILITARY ACTIVITY: Enter the name of the departmental project office or laboratory sponsoring *(paying for)* the research and development. Include address.

13. ABSTRACT: Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as *(TS)*, *(S)*, *(C)*, or *(U)*.

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

14. KEY WORDS: Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, roles, and weights is optional.

UNCLASSIFIED
Security Classification