Evolutionary Programming

Introduction

Simulated evolution is the process of duplicating certain aspects of the evolutionary system in the hopes that such an undertaking will produce artificially intelligent automata that are capable of solving problems in new and undiscovered ways, and in the execution of such an inquiry they hope to discover a deeper understanding of the very organization of intellect.

The basis of this approach is the humble admittance that while humans appear to be very intelligent creatures, there is no reason to purport that we are the most intelligent creatures that could possibly exist.

Historical EP perspective

- EP aimed at achieving intelligence
- Intelligence was viewed as adaptive behaviour
- Prediction of the environment was considered a prerequisite to adaptive behaviour
- Thus: capability to predict is key to intelligence

Table of contents

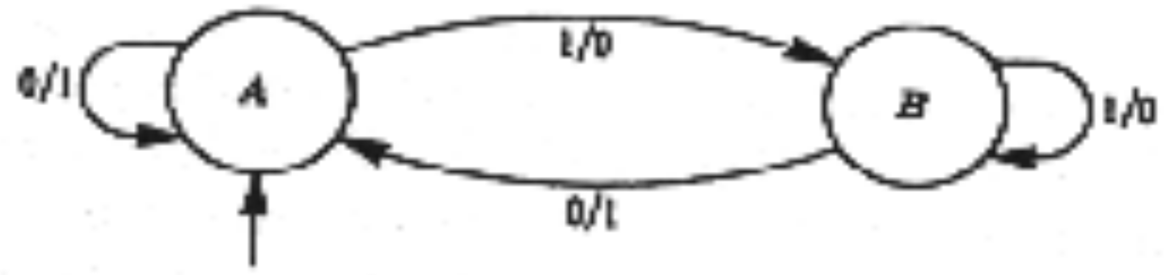
Theory

“Intelligent behavior is a composite ability to predict one’s environment coupled with a translation of each prediction into a suitable response in light of some objective” (Fogel et al., 1966, p. 11)

Success in predicting an environment is a prerequisite for intelligent behavior.

Theory

Let us consider the environment to be a sequence of symbols taken from a finite alphabet. The task before us is to create an algorithm that would operate on the observed indexed set of symbols and produce an output symbol that agrees with the next symbol to emerge from the environment.



Environment:	1	0	1	1	1	0	0	1	1	1	0	1	
FSM Predicts:		0	1	0	0	0	1	1	0	0	0	1	
Current State:		A	B	A	B	B	B	A	A	B	B	B	A

Theory

The basic procedure is as follows:

A collection of algorithms makes up the initial population, and they are graded based on how well they predict the next symbol to come out after being fed the given environment. The ones that receive a grade above some threshold level are retained as parents for the next iteration, the rest are discarded.

These offspring are then judged by the same criteria as their parents, and the process continues until an algorithm of sufficient quality is achieved or the given time lapse period expires.

Theory

The machines can be judged in a variety of ways. We could judge a machine based on whether or not it predicted the next symbol correctly, one at a time, or we could first expose the machine to a number of symbols taken from the environment, then let it guess. Typically these judgments also tend incorporate considerations for maintaining efficiency by penalizing complex machines.

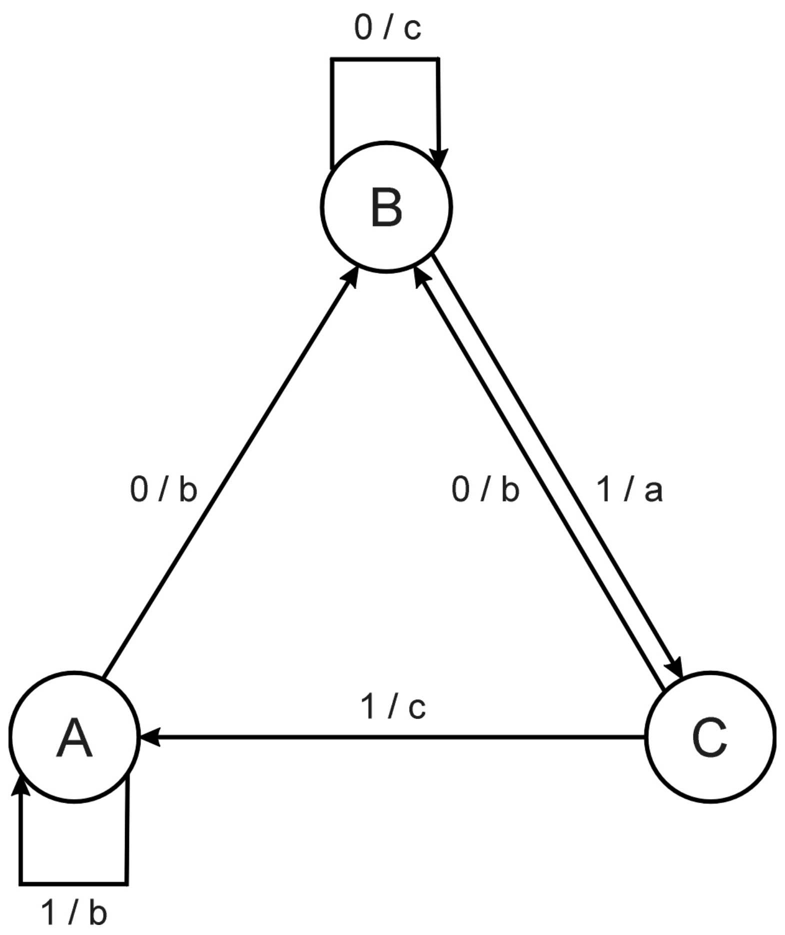
The recall length is the term used to describe how many symbols we expose the machine to before it has to make it's prediction.

Prediction by finite state machines

- Finite state machine (FSM):
  - States S
  - Inputs I
  - Outputs O
  - Transition function $\delta : S \times I \rightarrow S \times O$
  - Transforms input stream into output stream
- Can be used for predictions, e.g. to predict next input symbol in a sequence

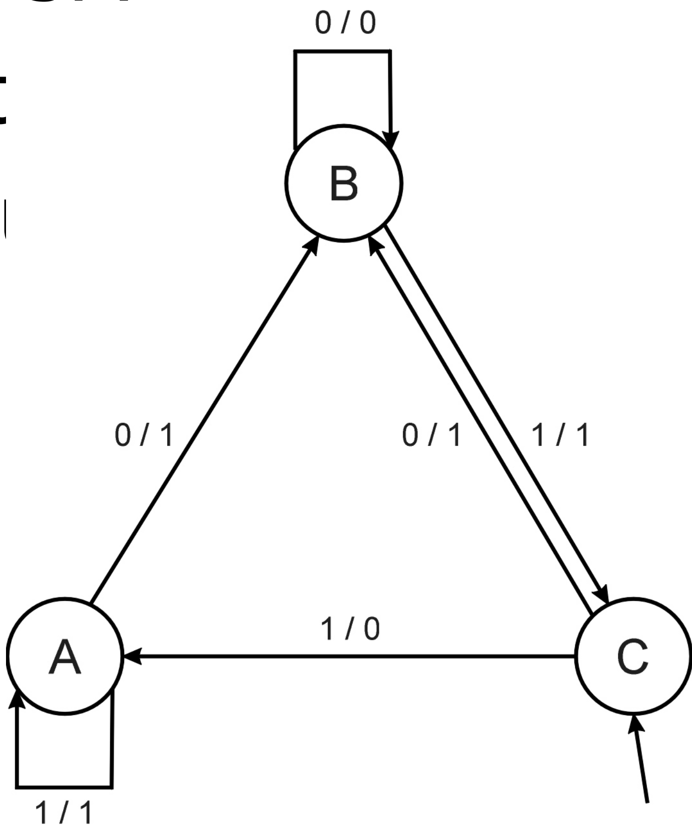
FSM example

- Consider the FSM with:
  - S = {A, B, C}
  - I = {0, 1}
  - O = {a, b, c}
  - δ given by a diagram



FSM as predictor

- Consider the following FSM
- Task: predict next input
- Quality: % of in$_{(i+1)}$ = o
- Given initial state C
- Input sequence 011101
- Leads to output 110111
- Quality: 3 out of 5



Introductory example: evolving FSMs to predict primes

- $P(n) = 1$ if n is prime, 0 otherwise
- $I = \boldsymbol{N} = \{1,2,3,\ldots, n, \ldots\}$
- $O = \{0,1\}$
- Correct prediction: $out_i = P(in_{(i+1)})$
- Fitness function:
  - 1 point for correct prediction of next input
  - 0 point for incorrect prediction
  - Penalty for “too much” states

1.2 Prediction Experiments

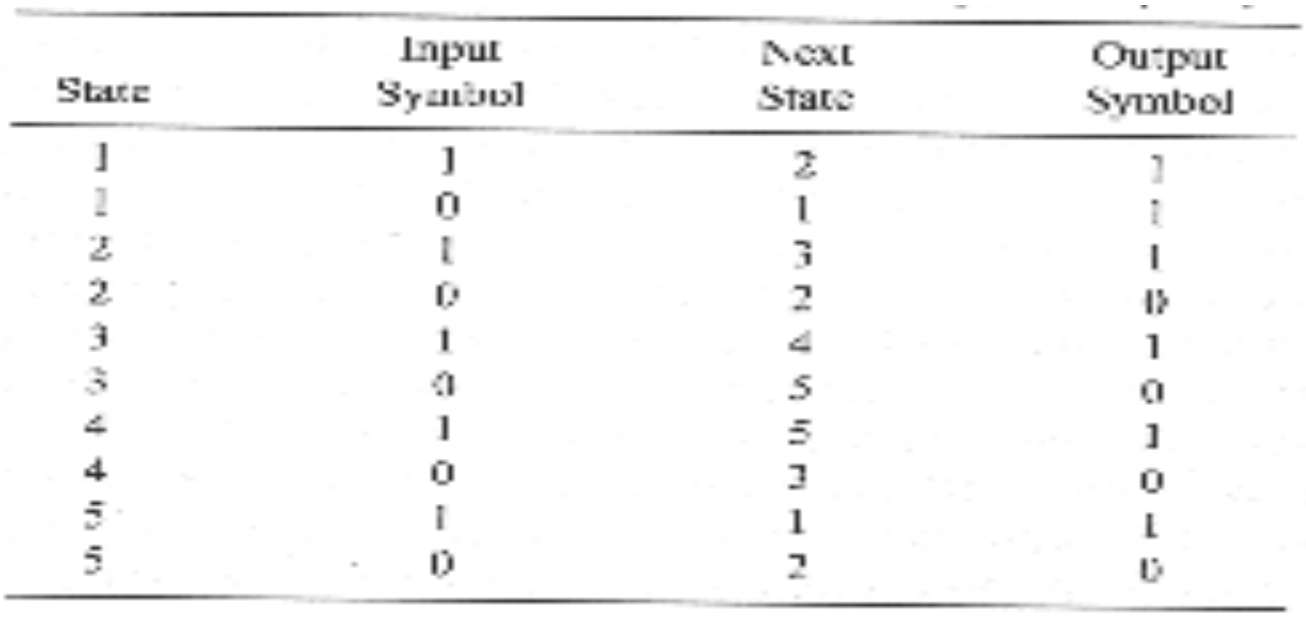
Prediction Experiments

In Fogel's Prediction Experiments, there is a given environment at the start, which is a series of symbols from our input alphabet. The initial machines, which are all identical, are run through the environment and judged based on how well they predict the symbols that follow. At the end the best three machines are kept and run through a series of mutations to create 3 more offspring. All 6 machines are then run through the same testing procedure, and the best 3 are chosen… and so on… {(P + C) selection.}

Every five iterations, the best machine is taken and told to predict the next symbol based on the last input symbol given, and the output given is taken and attached to the environment string.

Prediction Experiments

The Fogel experiments were done using the 5-state machine in Table 1.1 as the initial machine (all of the seed machines were a copy of this one).



State	Input Symbol	Next State	Output Symbol
1	1	2	1
1	0	1	1
2	1	3	1
2	0	2	0
3	1	4	1
3	0	5	0
4	1	5	1
4	0	3	0
5	1	1	1
5	0	2	0

Table 1.1

Prediction Experiments

The first four experiments were used to demonstrate the sensitivity of the procedure's capability to predict symbols in the sequence as a function of the types of mutation that were imposed on the parent machines. The environment used was the repeating pattern (101110011101)*. These initial experiments have no penalty for complexity (why a penalty for complexity? Well because huge machines would simply develop that are nothing but the sequence of symbols we input! This is not the desired end!) and only a single mutation was applied to each parent to derive it's offspring. Mutation was one of these 5 :

Add a state

Delete a state

Randomly change a next state link

Randomly change the start state

Change the start state to the 2nd state assumed under available experience.

Prediction Experiments

Figure 1.5 shows the results from four experiments in terms of the percent correct as a function of the number of symbols experienced in the environment. Several thousand generations were undertaken, and each of the final machines grew to between 8 and 10 states.

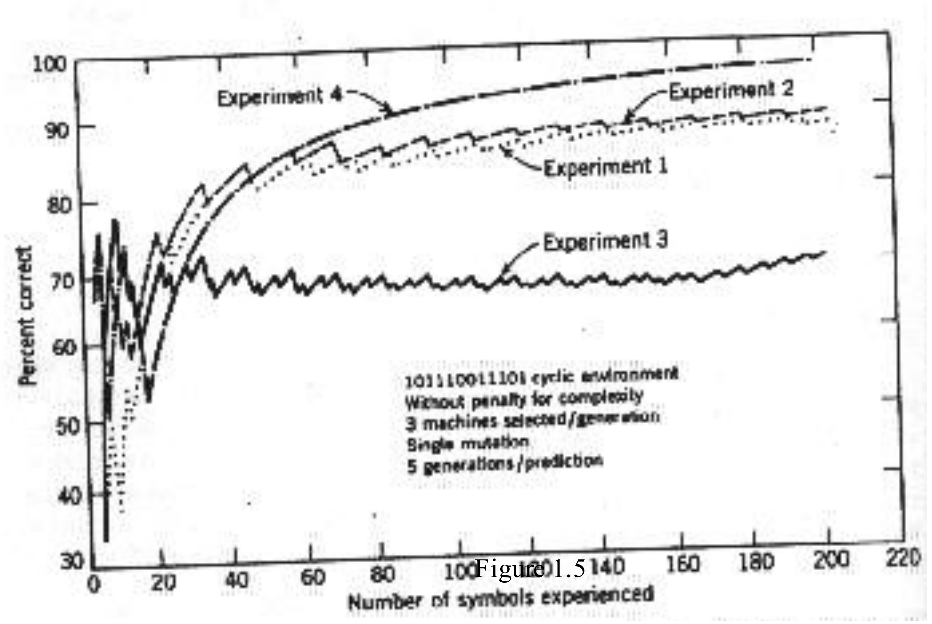


Figure 1.5

Prediction Experiments

In experiment 4 a series of perfect predictor-machines were found after the 19th symbol of experience. Poorest prediction occurred in experiment 3, but even this machine showed a remarkable tendency to predict well after the first few iterations of the environment string.

The 1st experiment is considered typical and will be used as the basis for comparison from now on.

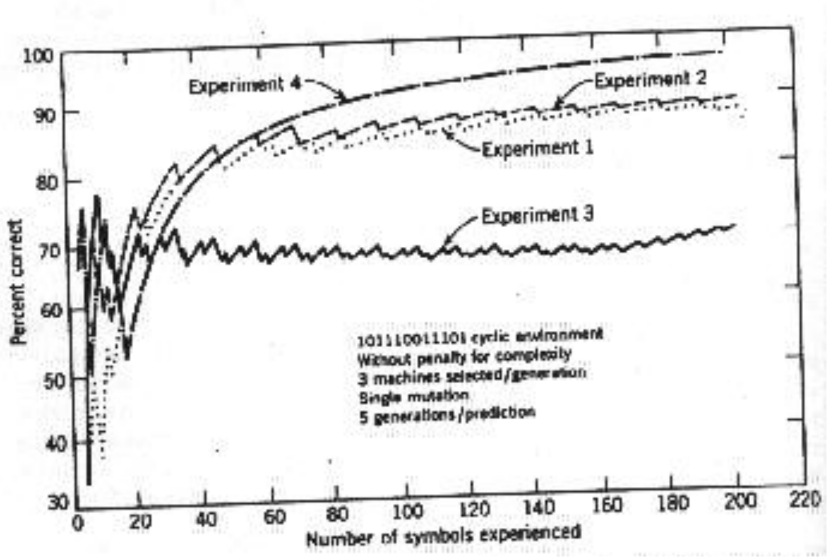


Figure 1.5

Prediction Experiments

Machine Complexity

The effect of imposing a penalty for machine complexity is shown in figure 1.6. The solid curve of experiment 5 represents experiment 1 duplicated with a penalty of 0.01 (or 1%) per state.

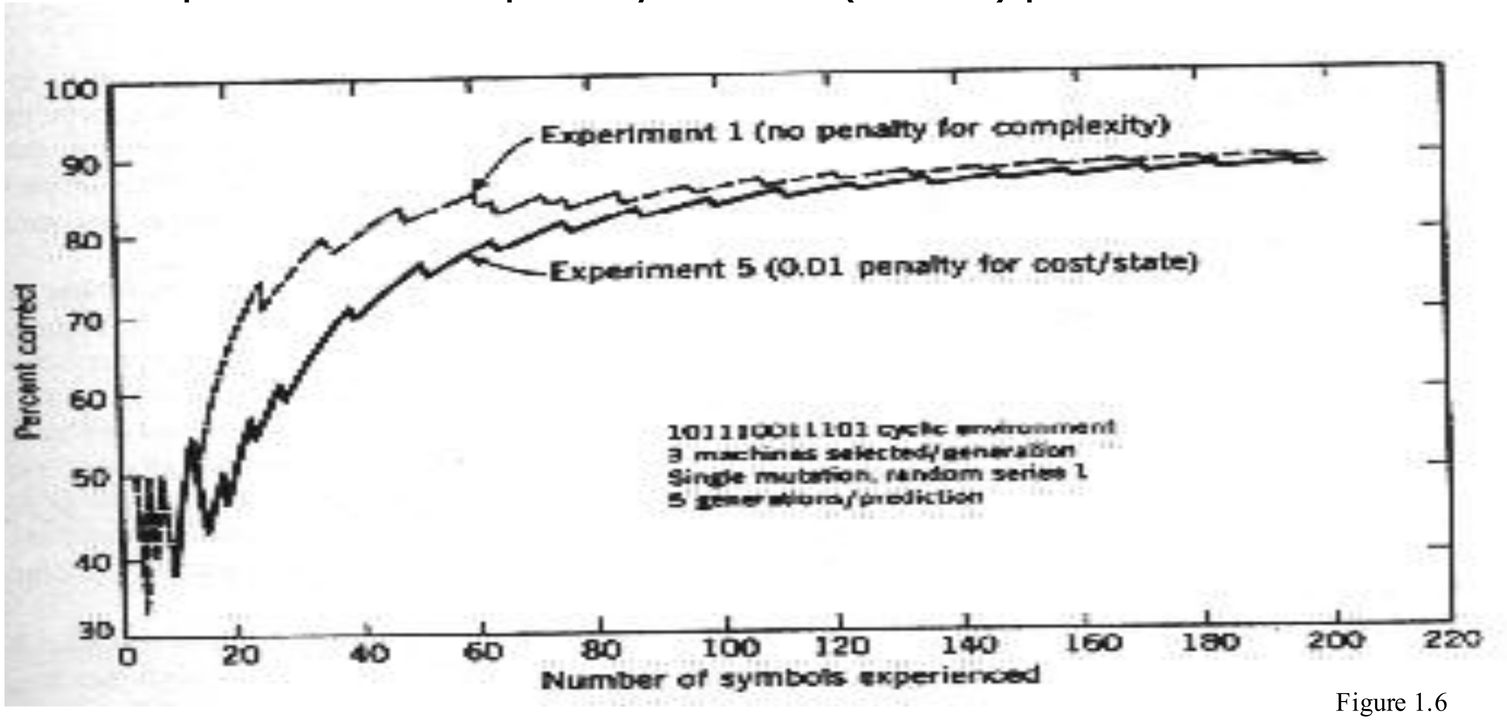


Figure 1.6

Prediction Experiments
Machine Complexity

The benefit of such a penalty can be seen in figure 1.7, which shows experiment 5 to have significantly less states, but as we can see in figure 1.6 the only time there is a significant difference in prediction capability is in the beginning.

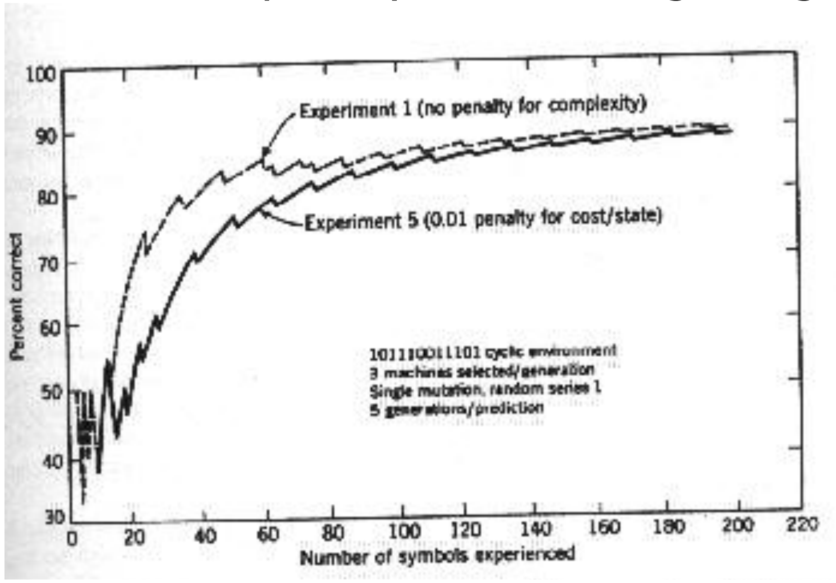


Figure 1.6

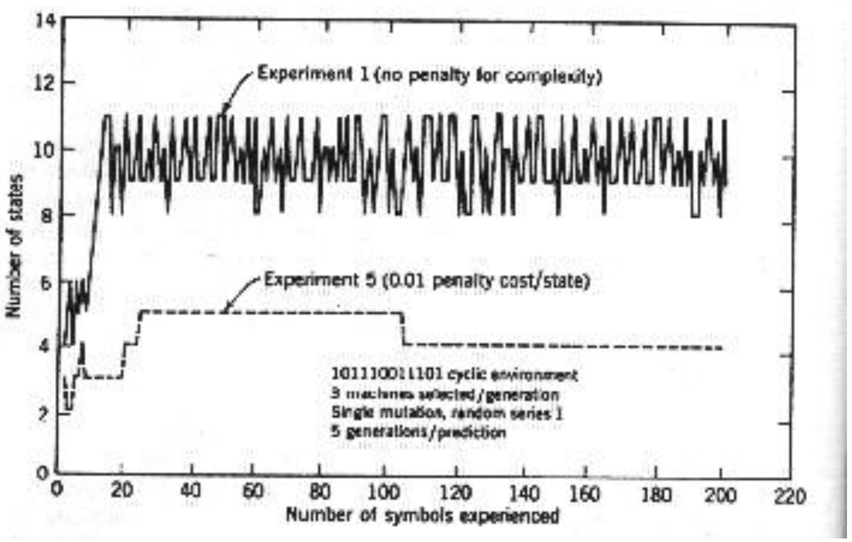


Figure 1.7

Prediction Experiments
Mutation Adjustments

It is reasonable to suspect that by increasing the probability of the ‘add-a-state’ mutation we might improve the prediction capability.

This is demonstrated in figure 1.9, where experiment 6 is a repetition of experiment 1 with the probability of the ‘add-a-state’ increased to 0.3 compensated by bringing the ‘delete-a-state’ down to 0.1. We can see that experiment 6 outperforms experiment 1.

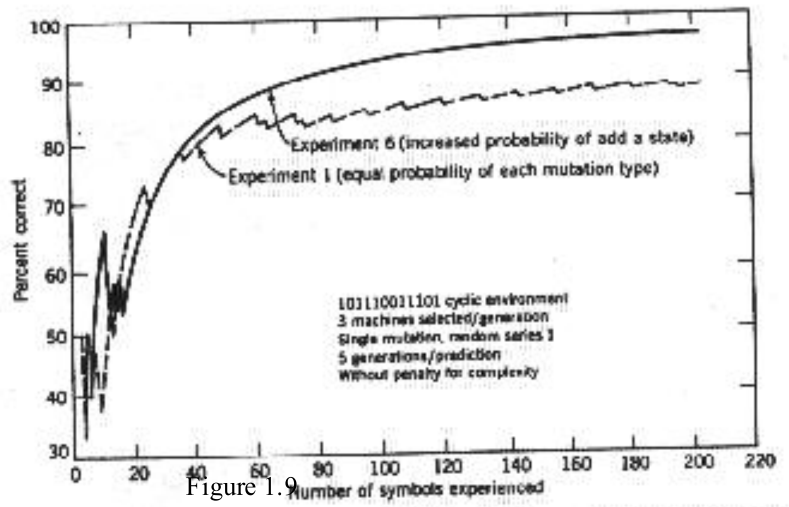


Figure 1.9

Prediction Experiments

Number of Mutations

The benefits of increasing the number of mutations per iteration is shown in figure 1.10, which shows experiments 1, 7, and 8 representing single, double, and triple mutation respectfully. The size of each of these machines is shown in figure 1.11.

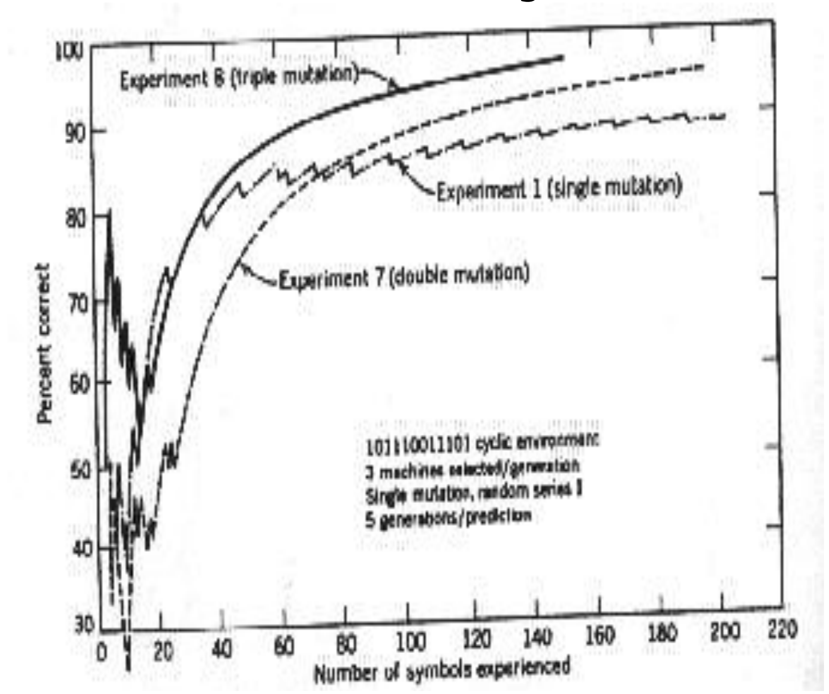


Figure 1.10

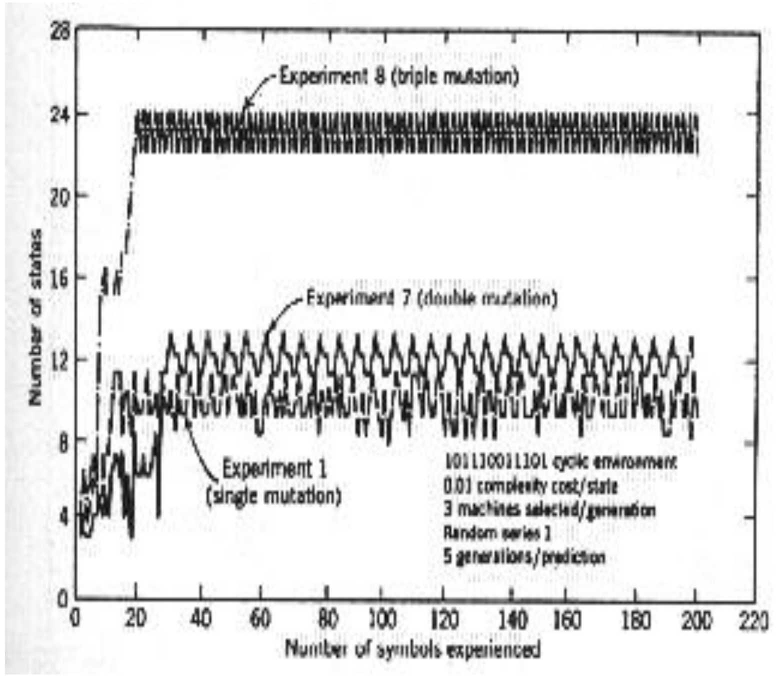


Figure 1.11

Prediction Experiments
Recall Length

In the case of a purely cyclic environment with no change to the input symbols, increasing the recall length provides for a larger sample size and an increased prediction rate.

In a noisy environment that has changes to the environment string it might be better to forget some past symbols

Prediction Experiments

Recall Length

Figure 1.12 shows the difference in recall lengths. During the initial sequence, the behavior appears quite random, but one can see that the longer recall length did exhibit faster learning of the cyclic environment.

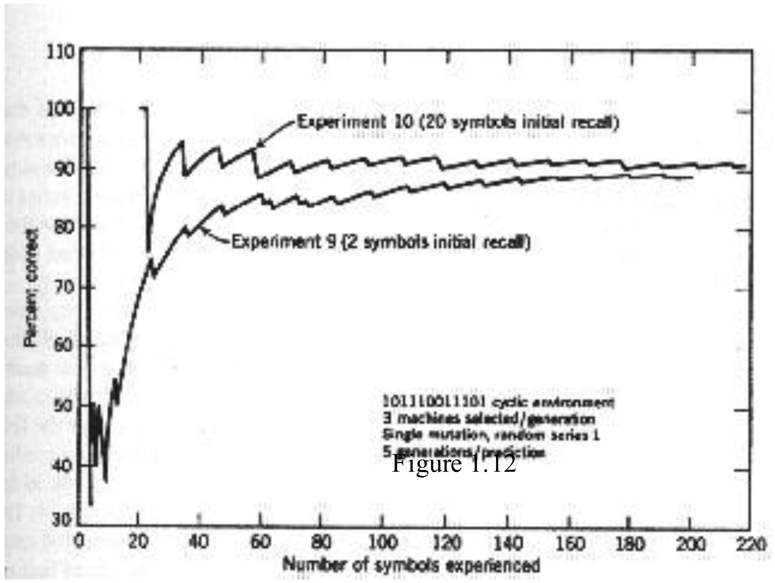


Figure 1.12

Prediction Experiments

Radical Change in Environment

Figure 1.13 and 1.14 demonstrate some interesting behavior. The solid line of Figure 1.13 demonstrates a normal evolutionary transition, but at symbol number 120 the environment undergoes a radical change. This change was the complete reversal of all the symbols in our environment.

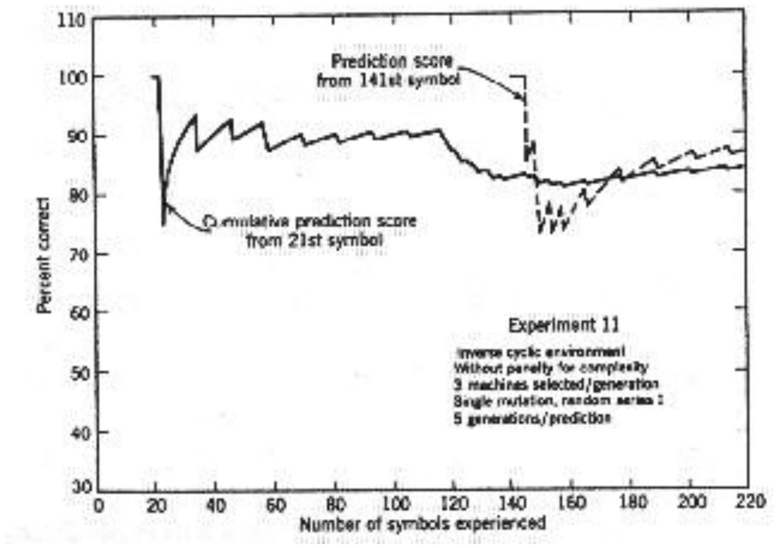


Figure 1.13

Prediction Experiments
Radical Change in Environment

One can see in figure 1.14 that it was at this point that the number of states shot through the roof as a great deal of “unlearning” had to take place.

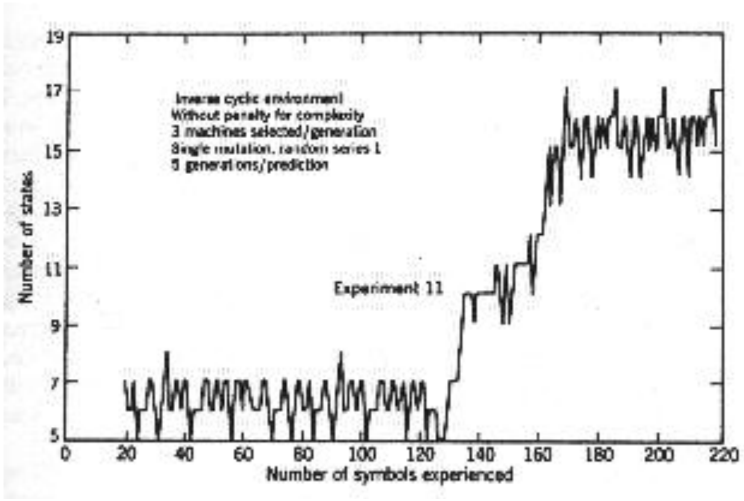


Figure 1.14

Prediction Experiments

Radical Change in Environment

The dotted line in figure 1.13 shows the comparison of machines that were not exposed to the radical change and instead started after it had already occurred. This score compares favorably with the first solid line when one considers that a machine is judged over the entire l

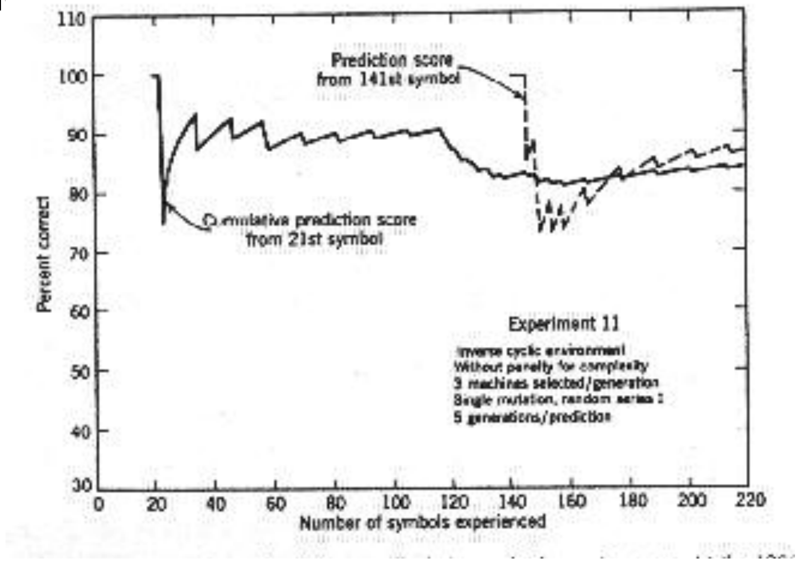


Figure 1.13

Evolutionary Computation

- Evolutionary Computation is the field of study devoted to the design, development, and analysis is problem solvers based on natural selection (simulated evolution).
- Evolution has proven to be a powerful search process.
- Evolutionary Computation has been successfully applied to a wide range of problems including:
  - Aircraft Design,
  - Routing in Communications Networks,
  - Tracking Windshear,
  - Game Playing (Checkers [Fogel])

Prediction Experiments

Predicting Primes

The most interesting of all these experiments is when they started to make the environment represent the appearance of prime numbers in an incremental count within the string.

For example, 01101010001, digits 2, 3, 5, 7, and 11 are all 1's.. which are all the prime numbers.

Prediction Experiments

Predicting Primes

We can see in figure 1.16 that experiment 15 ended up predicting the prime numbers quite well towards the end, and we can see in figure 1.17 that it ended up with very few states. This is easily understood when one notices that the higher we get into the environment string the less frequent prime numbers become.

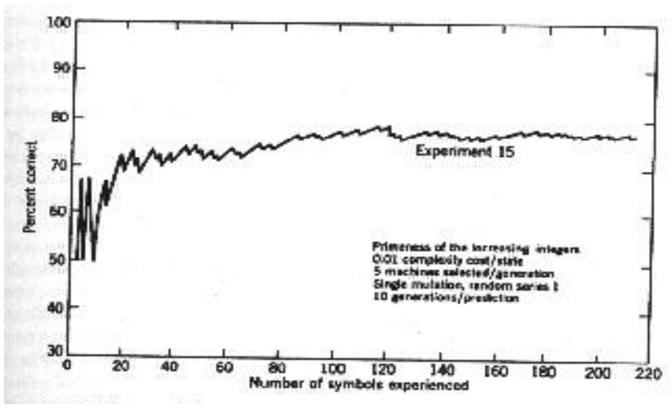


Figure 1.16

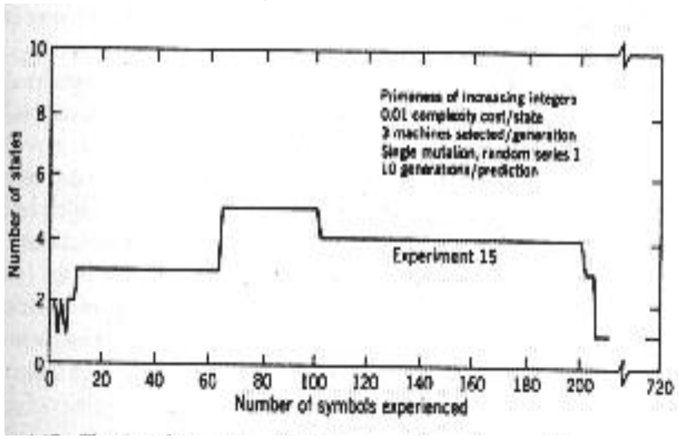


Figure 1.17

The results were obtained with a penalty for complexity of 0.01 per state, 5 machines per evolutionary iteration, and 10 rounds of mutation/selection before each prediction of a new symbol.

Prediction Experiments
Predicting Primes

To make things more interesting they increased the length of recall and gave a bonus for predicting a rare event. So the score given for predicting a 1 was the number of 0's that preceded it and the score given for predicting a 0 was the number of 1's that preceded it. One can see that predicting a 1 is much more valuable than predicting a 0.

Analysis of the results showed that the machines quickly "learned" to recognize numbers divisible by 2 and 3 as not prime, and some hints towards an increased tendency to predict multiples of 5's as not prime.

Thoughts

Evolutionary programming is not so much about programming, its more about the evolution of automaton.

The interesting thing, compared to some of the genetic algorithms, is that now you don't just have a bit string that encodes parameters, but you have to encode the initial state, the transition table, and the alphabet, and then you have to come up with problem specific mutations, or genetic mutators…This is nothing like the recombination mutation we saw in the last presentation.

Example application: evolving checkers players (Fogel'02)

- Neural nets for evaluating future values of moves are evolved
- NNs have fixed structure with 5046 weights, these are evolved + one weight for "kings"
- Representation:
  - vector of 5046 real numbers for object variables (weights)
  - vector of 5046 real numbers for σ's
- Mutation:
  - Gaussian, lognormal scheme with σ-first
  - Plus special mechanism for the kings' weight
- Population size 15

1.3 Pattern Recognition and Classification

Pattern Recognition and Classification

The key to understanding a sequence of foreign symbols is to try and find a recognizable pattern within them. If the symbols have no pattern, it is assumed to be random, in contrast if we can turn out a good prediction score it may reveal the presence of an unchanging signal. Variability in prediction score means the data may contain a message. If we CAN demonstrate a good prediction score, the question arises : what is the nature of the signal? Well, the state machine that achieved the acceptable score is a pretty good description in itself.

Pattern Recognition and Classification

So how well do these state machines describe the signal? And how well can they emulate human thought? Can they recognize and classify patterns in the same manner as a human operator?

Pattern Recognition and Classification

The following experiment was conducted. A series of broadband signals were generated and then dumbed down so as to be expressed in an 8-symbol alphabet, allowing them to be input into a computer program that would evolve to predict their behavior. They were generated with the goal of creating 4 sets of 4 signals that held basic similarities, such as the number of peaks and valleys and their locations being roughly the same.

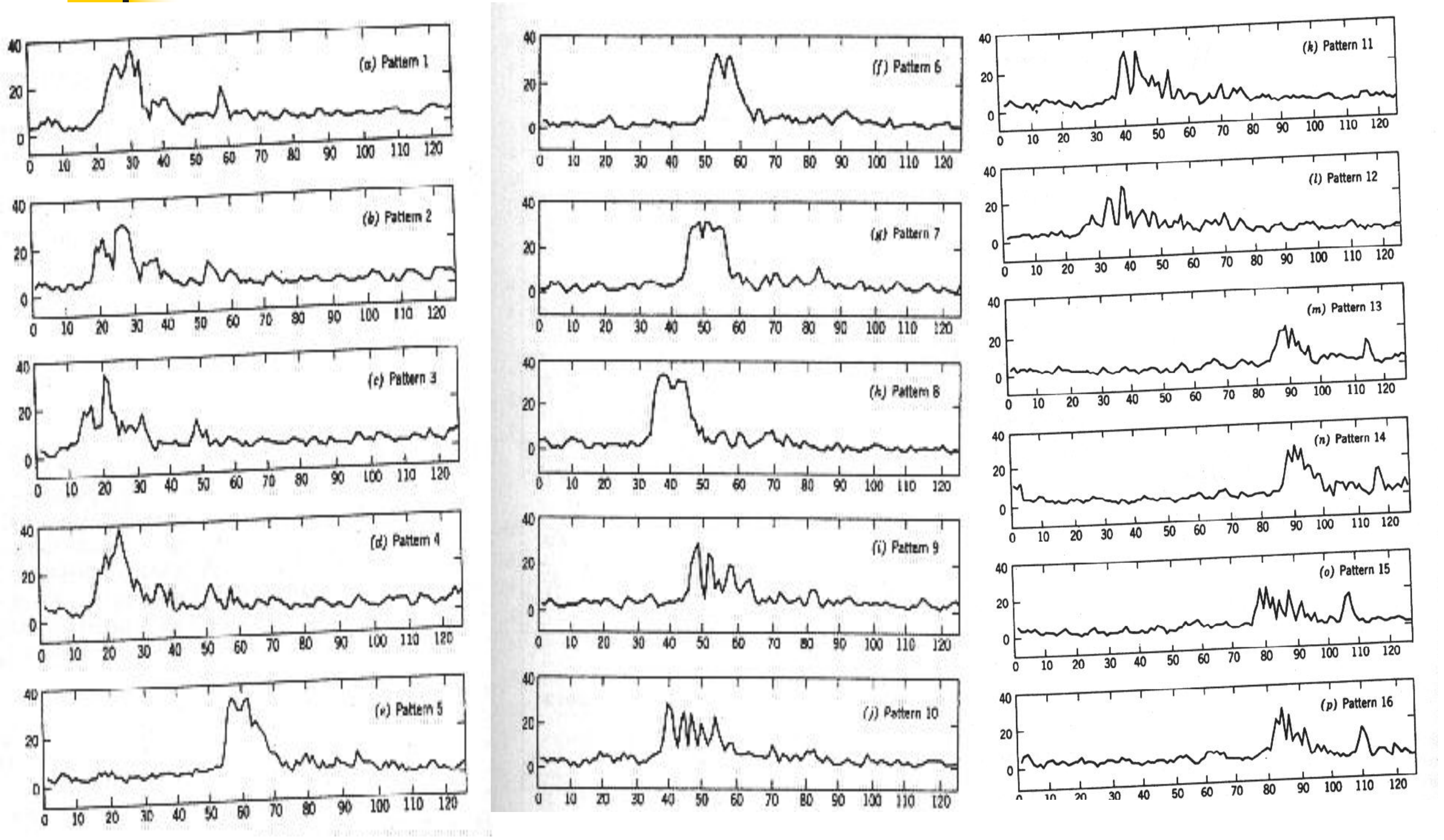
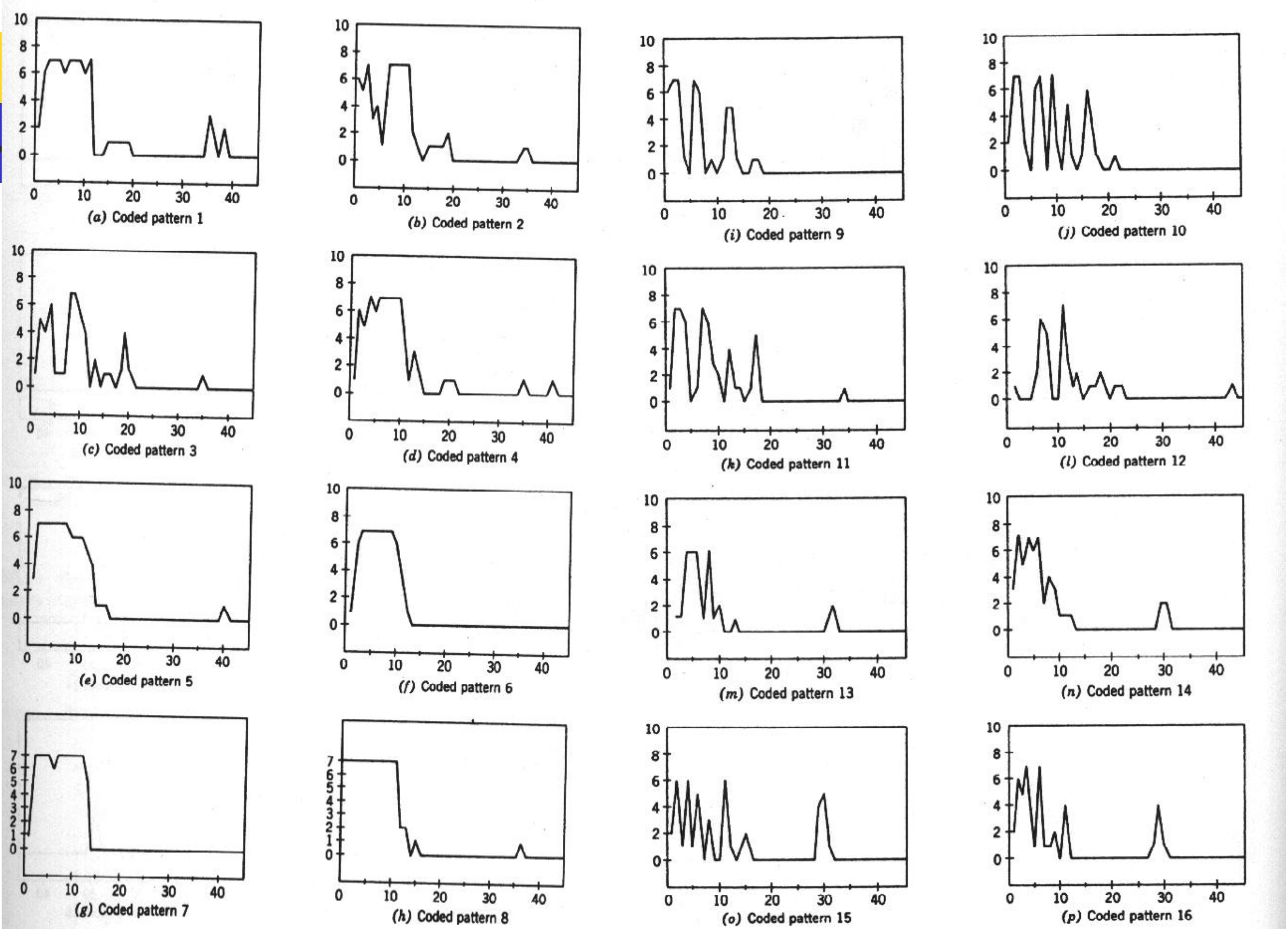

(a) Pattern 1
(b) Pattern 2
(c) Pattern 3
(d) Pattern 4
(e) Pattern 5
(f) Pattern 6
(g) Pattern 7
(h) Pattern 8
(i) Pattern 9
(j) Pattern 10
(k) Pattern 11
(l) Pattern 12
(m) Pattern 13
(n) Pattern 14
(o) Pattern 15
(p) Pattern 16
0
20
40
0 10 20 30 40 50 60 70 80 90 100 110 120

Figure 1.21


(a) Coded pattern 1

(b) Coded pattern 2

(i) Coded pattern 9

(j) Coded pattern 10

(c) Coded pattern 3

(d) Coded pattern 4

(k) Coded pattern 11

(l) Coded pattern 12

(e) Coded pattern 5

(f) Coded pattern 6

(m) Coded pattern 13

(n) Coded pattern 14

(g) Coded pattern 7

(h) Coded pattern 8

(o) Coded pattern 15

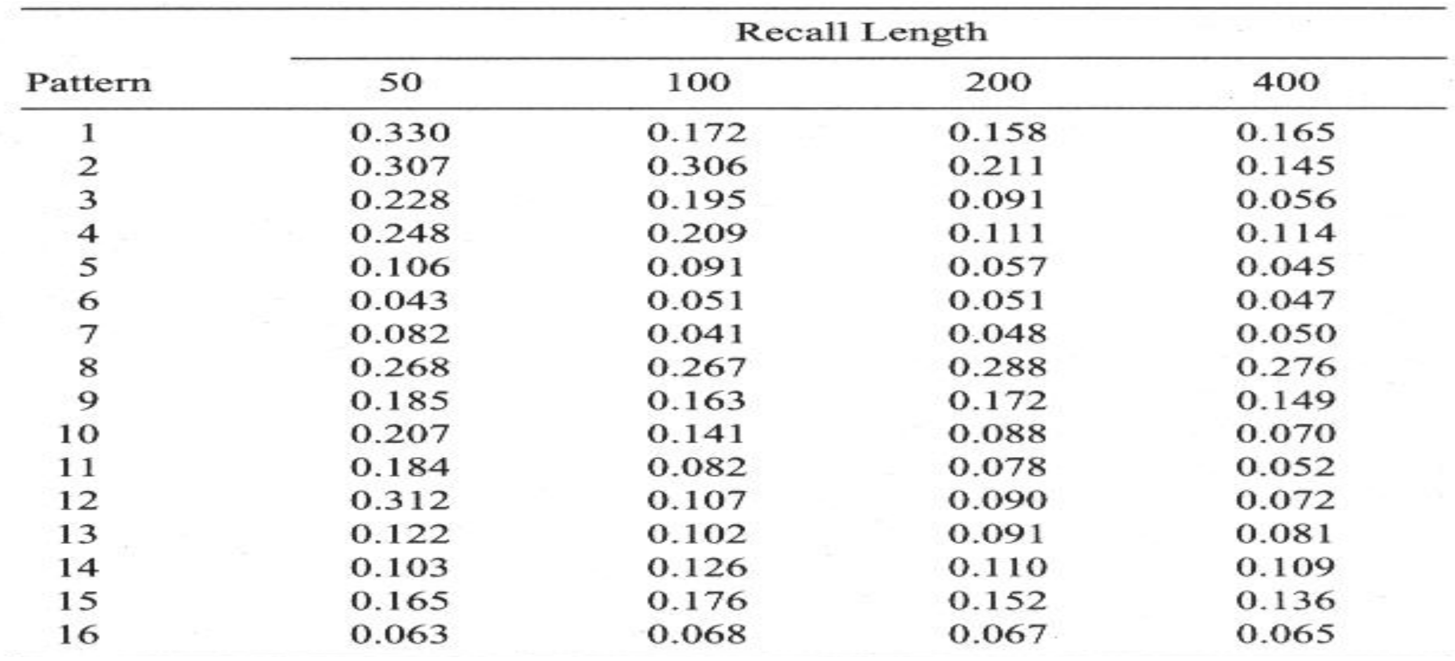
(p) Coded pattern 16

Pattern Recognition and Classification

An eight-symbol evolutionary program was used to predict each next symbol in an unending repetition of each of these patterns. There was no penalty for complexity, and 10 generations prior to each prediction. There was also a “magnitude of the difference” error cost matrix specification of the goal.

Pattern Recognition and Classification

Table 1.2 indicates the average prediction error rate of these evolutionary programs applied to their own signal after the first 50, 100, 200, and 400 predictions. It can be seen that the greatest amount of “learning” occurred in the early stages of development.



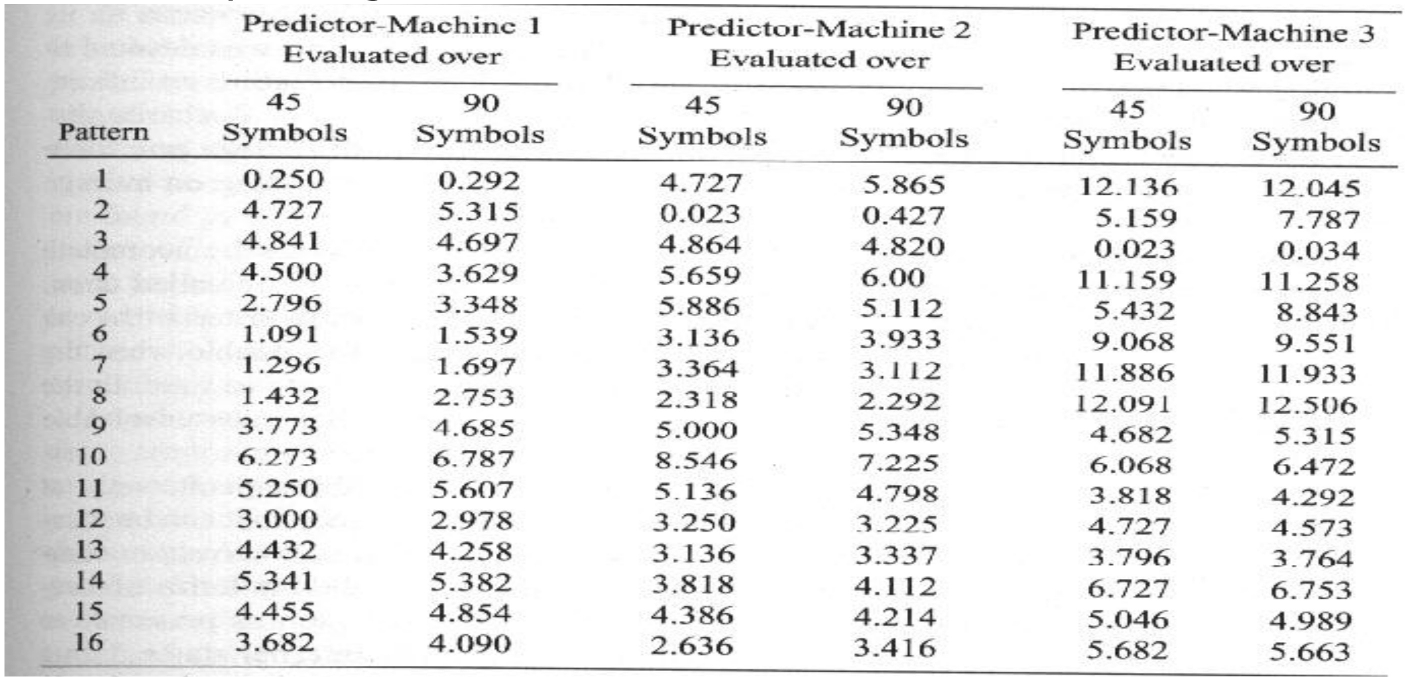
	Recall Length			
Pattern	50	100	200	400
1	0.330	0.172	0.158	0.165
2	0.307	0.306	0.211	0.145
3	0.228	0.195	0.091	0.056
4	0.248	0.209	0.111	0.114
5	0.106	0.091	0.057	0.045
6	0.043	0.051	0.051	0.047
7	0.082	0.041	0.048	0.050
8	0.268	0.267	0.288	0.276
9	0.185	0.163	0.172	0.149
10	0.207	0.141	0.088	0.070
11	0.184	0.082	0.078	0.052
12	0.312	0.107	0.090	0.072
13	0.122	0.102	0.091	0.081
14	0.103	0.126	0.110	0.109
15	0.165	0.176	0.152	0.136
16	0.063	0.068	0.067	0.065

Pattern Recognition and Classification

Each evolved machine was a characterization of the signal in which it developed, this is obvious. One might think it is also obvious that we recognize similarities in the signals through similarities in the machines, but this is not such an easy task since these machines can often grow to be very complex, and what method would you use to make such a comparison? It is much more natural to accomplish the comparison by allowing the evolved machines to attempt a prediction of the OTHER, similar signals. The similarity between patterns should be demonstrated by the similarity in prediction scores.

Pattern Recognition and Classification

Well, table 1.3 shows the results of such a comparison, and things did not turn out the way we had hoped. As was expected, each machine predicted it's own signal very well, but the remaining scores showed that none could classify the signals in the desired manner.



	Predictor-Machine 1 Evaluated over		Predictor-Machine 2 Evaluated over		Predictor-Machine 3 Evaluated over	
Pattern	45 Symbols	90 Symbols	45 Symbols	90 Symbols	45 Symbols	90 Symbols
1	0.250	0.292	4.727	5.865	12.136	12.045
2	4.727	5.315	0.023	0.427	5.159	7.787
3	4.841	4.697	4.864	4.820	0.023	0.034
4	4.500	3.629	5.659	6.00	11.159	11.258
5	2.796	3.348	5.886	5.112	5.432	8.843
6	1.091	1.539	3.136	3.933	9.068	9.551
7	1.296	1.697	3.364	3.112	11.886	11.933
8	1.432	2.753	2.318	2.292	12.091	12.506
9	3.773	4.685	5.000	5.348	4.682	5.315
10	6.273	6.787	8.546	7.225	6.068	6.472
11	5.250	5.607	5.136	4.798	3.818	4.292
12	3.000	2.978	3.250	3.225	4.727	4.573
13	4.432	4.258	3.136	3.337	3.796	3.764
14	5.341	5.382	3.818	4.112	6.727	6.753
15	4.455	4.854	4.386	4.214	5.046	4.989
16	3.682	4.090	2.636	3.416	5.682	5.663

Table 1.3

Pattern Recognition and Classification

It is evident that the predictor machines recognize similarity in a much different way than do humans. A human operator would simply look at the signals and note the number of peaks and valleys and their relative position and magnitude, making the comparison a trivial task. But there is no demand that the evolutionary program emulate human behavior in performing the same task. According to Fogel, it is this very constraint that has limited the advancement of AI in the past 30 years.

Control System Design

So far we’ve looked at such problems as detection (Is there a signal?) discrimination (if so, what is the signal?), recognition (has the signal been seen before?), classification (if not, which of a set of signals is it most like).

But almost all of these are of interest only in that they might precede steps towards a solution of the problem of control.

Control System Design

So what is this problem of control?

Let us define a system as a *plant*. This could be any system, be it a computer program, another state machine, or a living organism. We have no idea what the nature of this system is, all we know is that given some input string it will punch out some output string.

The problem of control is the attempt to understand such a system. We want to be able to tell the plant what to do and have it achieve some desired result or goal.

Classical Control

- The Mathematicians Approach
  - Rigidly Modeled System
- Software does what it is told
  - Intelligence comes from the Designer

Intelligent Control

- The Lazymans Approach
  - System not Rigidly Modeled
- Software does what it wants to
  - Intelligence comes from the Software

Open-Loop Control System

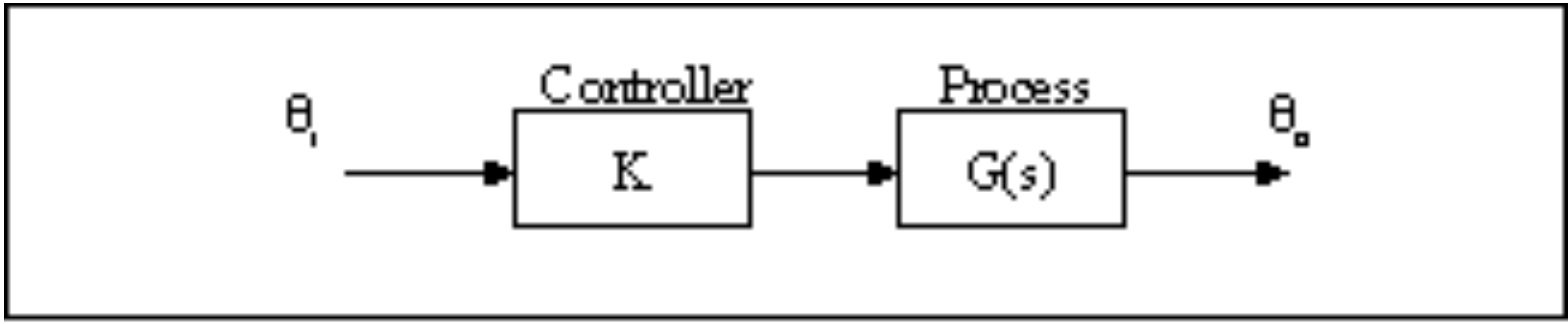


Figure 2.1

Closed-Loop Control System

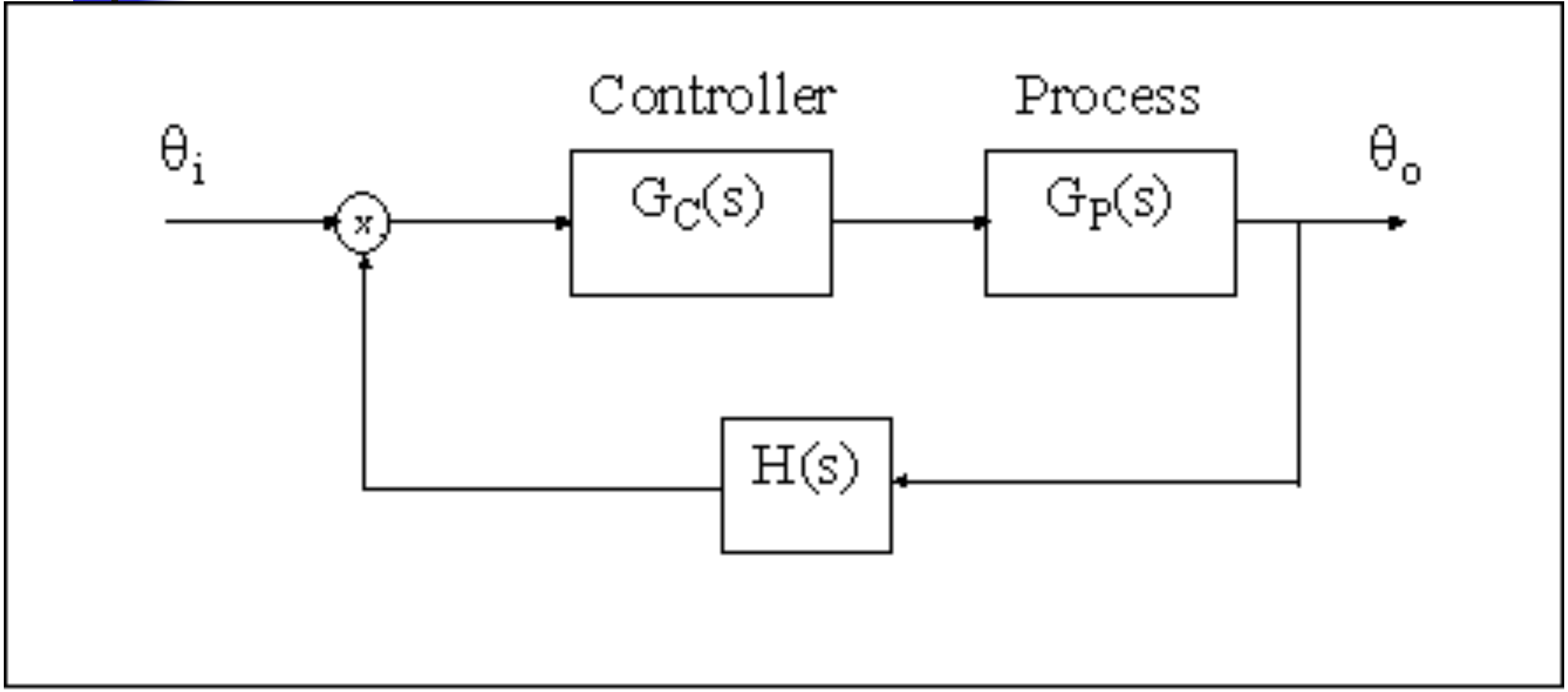
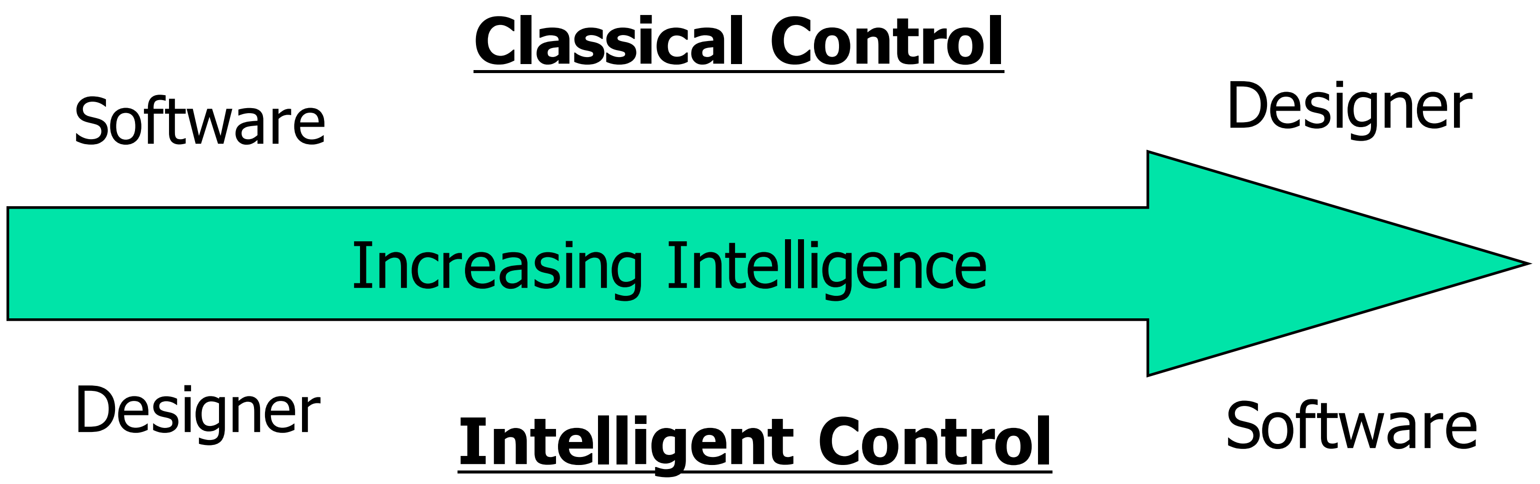


Figure 1

Shifting Intelligence



System Modeling

First-Order System: $$G(s) = \frac{1}{Ts+1}$$

Second-Order System: $$G(s) = \frac{1}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

Classical Control Examples

- PID Control
- Optimal Control
- Discrete-Event Control
- Hybrid Control

PID Control

- Proportional Control
  - Pure gain adjustment acting on error signal
- Integral Control
  - Adjust accuracy of the system
- Derivative Control
  - Adjust damping of the system

Control System Design

But if we don't understand anything about the nature of the system, and only have an output that was spewed out by the plant on some given input, how can we possibly hope to be able to control such a system, and be able to tell it what to do?

We use evolutionary programming.

Control System Design

How do we use evolutionary programming to solve the problem of control? The process is as follows :

1. Create a state machine that you believe best describes the plant, but this initial machine is actually not very relevant. In theory, it could be anything, but we should attempt to emulate the plant as close as we can.

2. We then give our newly created machine the sequence of input symbols that was given to our original plant, and judge it based on how well it could predict the actual output that was given by the plant.

Control System Design

3. We continually evolve the machine to become a perfect predictor of the plant, this meaning that the machine will spit out the same output as the plant when they are both given some input sequence.

4. Now, if we want to control the plant, we need to determine the input string that will achieve our desired end. To do this we simply look at our state machine and determine the input symbols that would be required to produce our desired output.

Control System Design

This is where the actual functionality of evolutionary programming comes in.

It allows us to develop a machine that will further allow us to understand some unknown system.

Intelligent Control Examples

- Fuzzy Logic Control
- Neural Network Control
- Genetic Programming Control
- Support Vector Machines
- Numerical Learning
- COMDPs - POMDPs

References

- *Genetic Programming FAQ,* 2002. http://www.cs.ucl.ac.uk/research/genprog/gp2faq/gp2faq.html
- Akbarzadeh, M.R., Kumbla, K., Tunstel, E., Jarnshidi, M., “Soft Comuting for Autonomous Robotic Systems”, *Computers and Electrivcal Engineering*, 2002: 26, pp. 5-32.
- Bojarczuk, C.C., Lopes, H.S., Freitas, A.A., “Genetic Programming for Knowledge Discovery In Chest-Pain Diagnosis”, *IEEE Engineering in Medicine and Biology Magazine,* 2000: 19, v. 4, pp. 38-44.
- Howley, B., “Genetic Programming of Near-Minimum-Time Spacecraft Attitude Maneuvers”, *Proceedings of Genetic Programming 1996,* Koza, J.R. et al. (Eds), MIT Press, 1996, pp. 98-109.
- Koza, J., “Genetic Programming as a Means for Programming Computers by Natural Selection“, 1994.
- Poli, R., “Genetic Programming for Image Analysis”, *Proceedings of Genetic Programming 1996,* Koza, J.R. et al. (Eds), MIT Press, 1996, pp. 363-368.
- Poli, R., “Parallel Distributed Genetic Programming”, *Technical* report, The University of Birmingham, School of Computer Science, 1996.
- Pujol, J.C.F., Poli, R., “Efficient Evolution of Asymetric Recurrent Neural Networks Using a Two-dimensional Representation”, 1997.
- Thompson, A., “Artificial Evolution in the Physical World”, *Evolutionary Robotics: From Intelligent Robots to Artificial Life*, Gomi, T. (Ed.), AAI Books, 1997, pp. 101-125.
- Zhang, B.T., Mühlenbein, H., “Genetic Programming of Minimal Neural Nets Using Occam’s Razor”, *Proc. Of 5th Int. Conf. On Genetic Algorithms,* 1993, pp. 342-349.