

THE UNIVERSITY OF MICHIGAN
COLLEGE OF LITERATURE, SCIENCE, AND THE ARTS
Computer and Communication Sciences Department

GENETIC ALGORITHMS
AS
FUNCTION OPTIMIZERS

by

bert D. Bethke

April 1978

THE UNIVERSITY OF MICHIGAN
ENGINEERING LIBRARY

Logic of Computers Group
Computer and Communication Sciences Department
Technical Report No. 212

with assistance from:

National Aeronautics and Space Administration
Grant No. NGG-1176
Langley Research Center
Hampton, Virginia

and

National Science Foundation Grant No.
MCS76-04297

enon
UMIL/341

This is the text of a talk given at
the ACM Computer Science Conference held
in Detroit, February 21-23, 1978.

The results I am about to present come primarily from two research projects in which I have participated. These results form the foundation for my thesis research. Genetic algorithms are the brainchild of my thesis committee chairman, Professor John H. Holland.

I would like to present a new, rather different approach to an old problem. This problem is easy to state but difficult to solve. It is: find the point at which a real-valued function takes its maximal value.

The function to be maximized is called the objective function and its domain, D , is the search space. We will assume that the search space is made up of n -tuples of real numbers. That is, the objective function is a real-valued function of n real variables.

The usual approach to such a difficult problem is to make some simplifying assumptions about the problem and hope that the solution obtained will be easily extended to solve the original problem. In particular, if the objective function is assumed to be differentiable or perhaps twice differentiable, it is possible to use calculus-based methods to find the optimum.

One such algorithm is the conjugate gradient method. (Actually, there are many minor variations on the basic method so this is a class of algorithms.) This method is essentially a modification of Newton's method.

The conjugate gradient method implicitly assumes that the objective function is approximately quadratic and works best, of course, on quadratic objective functions. It also works well on other convex, unimodal objective functions. In fact, it actually works reasonably well on most differentiable, unimodal objective functions. But it fails miserably when

the objective function is not unimodal or not sufficiently smooth.

Consider the following bi-modal objective function. (See figure 1.) If the conjugate method is applied with a starting value larger than 7.5, it will converge to the optimum. But if the starting point is less than 7.5, it will converge to the false peak.

There seems to be no easy way to "fix-up" the conjugate gradient method to handle multi-modal and non-differentiable objective functions. So we need a different approach.

Instead of basing our search strategy on the geometry of quasi-quadratic functions, let's make the simplifying assumption that the search space contains regions of good objective function values and regions of poor values. Let's represent the points in the search space as binary strings and let's look for good bit patterns.

For the bi-modal objective function shown here, we may choose to represent points in the interval from zero to sixteen by 12 bit binary numbers with the binary point assumed to be between the 4th and 5th bits. If we need more accuracy or a wider range of values, we may use more bits and/or a different code such as a "floating point" representation.

Naturally, we wish to concentrate our exploration efforts on the good regions of the search space. One class of simple algorithms which does this is genetic algorithms.

A genetic algorithm maintains a collection (population) of several strings and explores the search space by generating successive populations which are distributed differently in the search space. Hopefully, the distribution changes so as to cluster about the optimum as the algorithm proceeds.

The principal way of generating a new string is by crossover: two

strings are broken at a randomly chosen point, the final segments are switched, and we have two new strings (figure 2).

The overall algorithm consists of choosing a random initial population, and then generating successive populations using the following basic cycle (see figure 3). Each string is replicated in proportion to its function value to generate an intermediate population. Pairs of strings are chosen at random from the intermediate population and combined using crossover to generate the new population. In addition, a small number of randomly chosen bits in the new population are mutated (changed). This serves as a source of variability and improves the search pattern of the algorithm.

To get some feeling for how a genetic algorithm explores the search space, let's look at the following typical run (figure 4). The objective function is the bi-modal objective function described earlier, and the strings are 12 bit binary numbers with the binary point assumed to be between the 4th and 5th bits.

The initial population consists of random points spread throughout the interval from 0 to 16. After one time through the basic cycle, the points are beginning to cluster about the two peaks. After two more times through the basic cycle, nearly all the points are near the true optimum.

Genetic algorithms can optimize a large class of objective functions. My thesis research addresses the problem of describing that class of functions. For the purposes of this talk, it is sufficient to say that genetic algorithms work for a much larger class of functions than conjugate gradient methods.

In comparing two optimizers, the traditional cost measure is the number of times the objective function must be evaluated or sampled. Here are some comparisons between genetic algorithms and conjugate gradient

methods (figures 5 and 6). The red curve represents the performance of the conjugate gradient method; the green curve represents the performance of the genetic algorithm--actually the average performance since the genetic algorithm is stochastic and produces slightly different results for different random number streams.

For unimodal objective functions, the conjugate gradient method very rapidly converges to the optimum while the genetic algorithm takes much longer. For a bi-modal objective, the conjugate gradient method converges to the wrong peak for about one-half the starting points resulting in the average performance shown by the solid red curve. The genetic algorithm performs the same as on the unimodal functions. For a multi-modal objective function of many variables, the conjugate gradient method rarely finds the true optimum. The genetic algorithm, on the other hand, rarely gets trapped on a false peak.

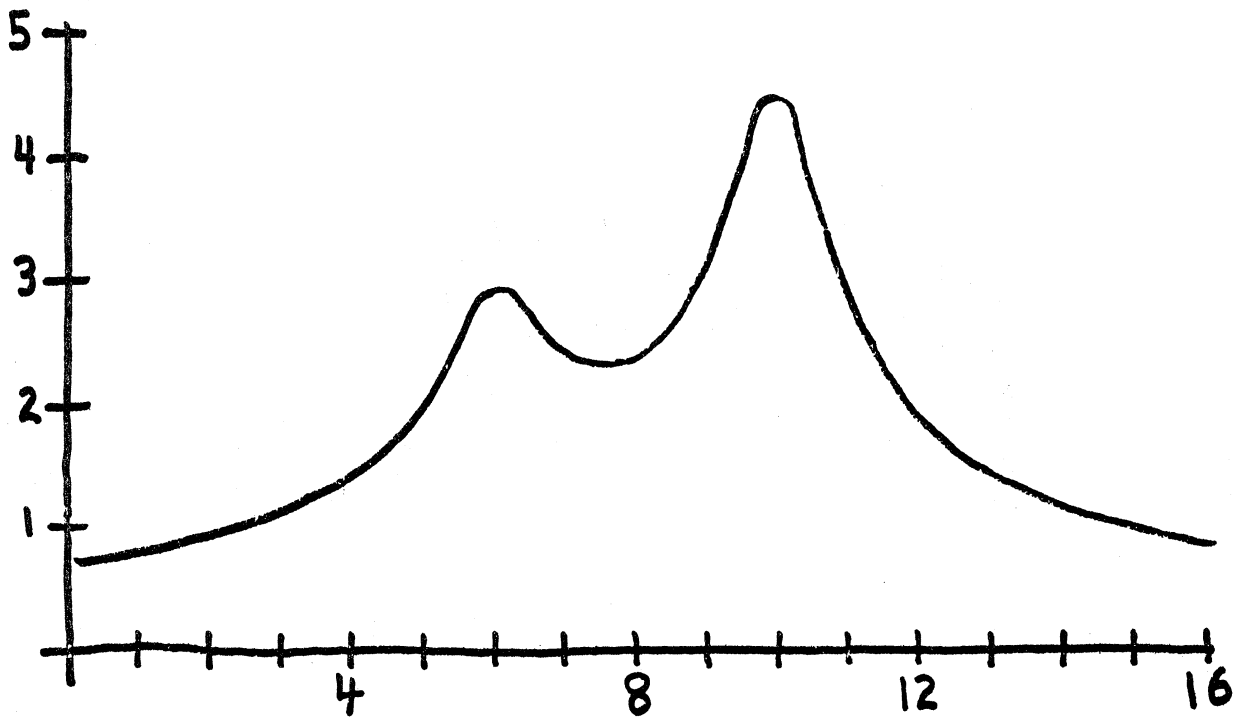
If the objective function is contaminated by a small amount of random noise, so that sampling the same point several times gives slightly different function values, then the conjugate gradient method behaves like random search. Genetic algorithms are insensitive to small amounts of noise and their performance is unchanged until the function values differ from the optimal value by less than the noise level. Performance curves for discontinuous objective functions are very similar to those for noisy objective functions. Also, note that genetic algorithms use less computational effort to generate each sample point. So comparisons based on CPU time would be more favorable to genetic algorithms.

In conclusion: If you know that the objective function has special properties which you can exploit, then use a special method like conjugate gradient or whatever is appropriate. If you don't know of any special

properties of the objective function, then you should use a method which is more general like genetic algorithms.

FIGURE 1

A BI-MODAL OBJECTIVE FUNCTION



$$f(x) = 6 - 2e^{\frac{-1}{|x-6|}} - 4e^{\frac{-1}{|x-10|}}$$

CONJUGATE GRADIENT METHOD
WILL FAIL IF STARTING POINT
IS TO THE LEFT OF 7.5.


FIGURE 2

CROSSOVER


INITIAL STRINGS

———— 10111000110
———— 010100101101

CHOOSE RANDOM BREAK POINT

 101110³00110
0101001³01101

EXCHANGE SEGMENTS

 101110³01101
0101001³00110

FINAL RESULTS

———— 10111001101
———— 010100100110

BASIC CYCLE

FIGURE 3

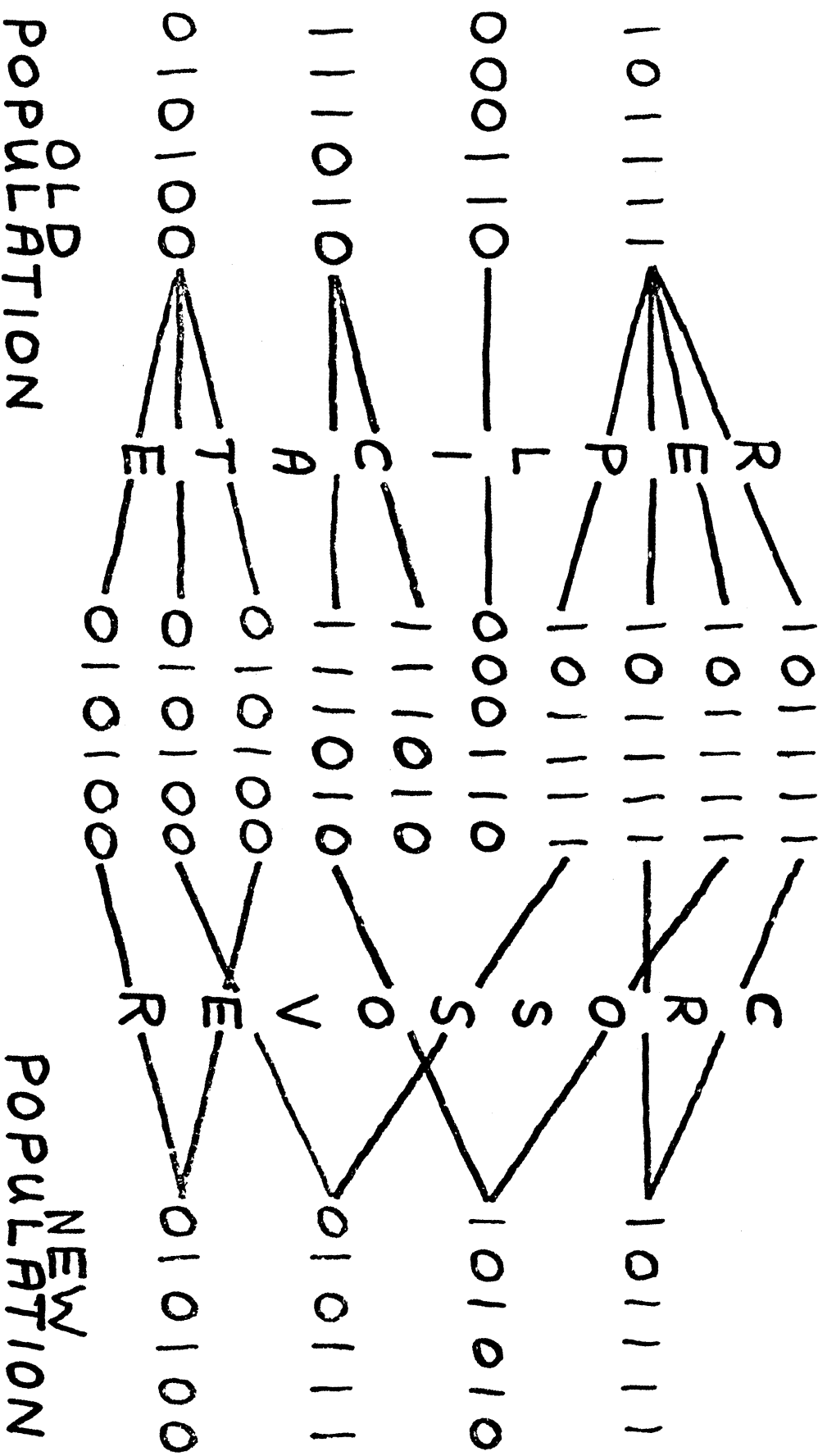


FIGURE 4

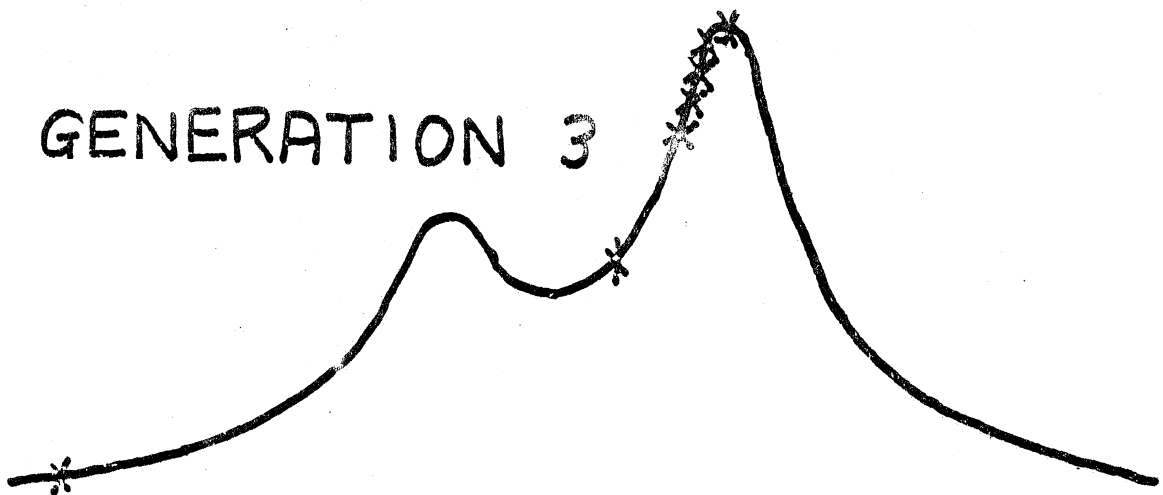
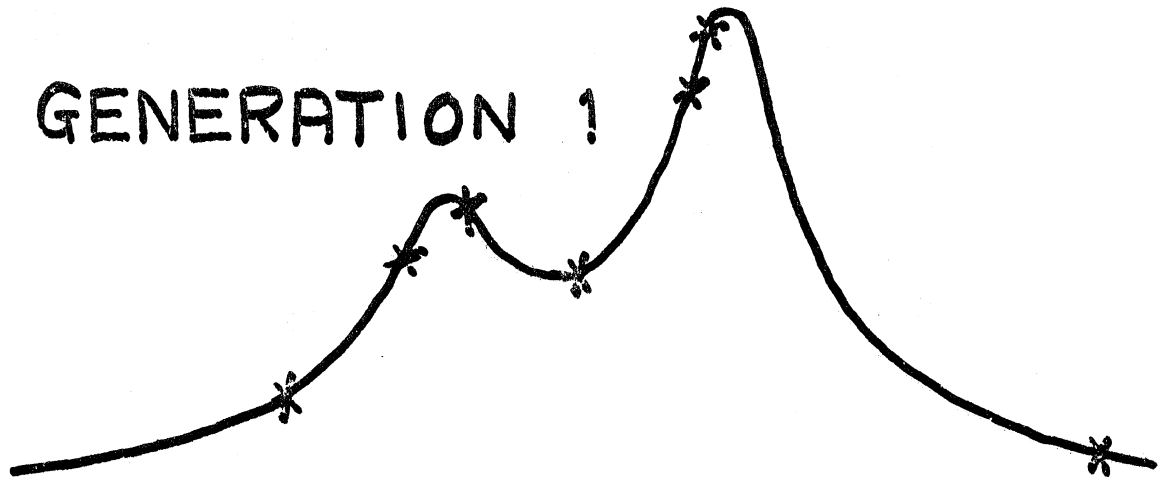
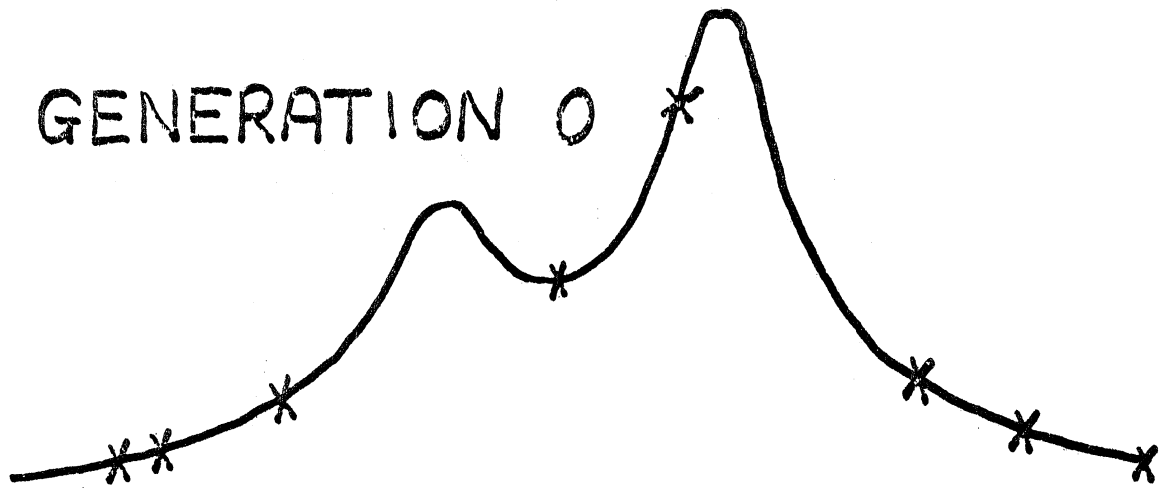
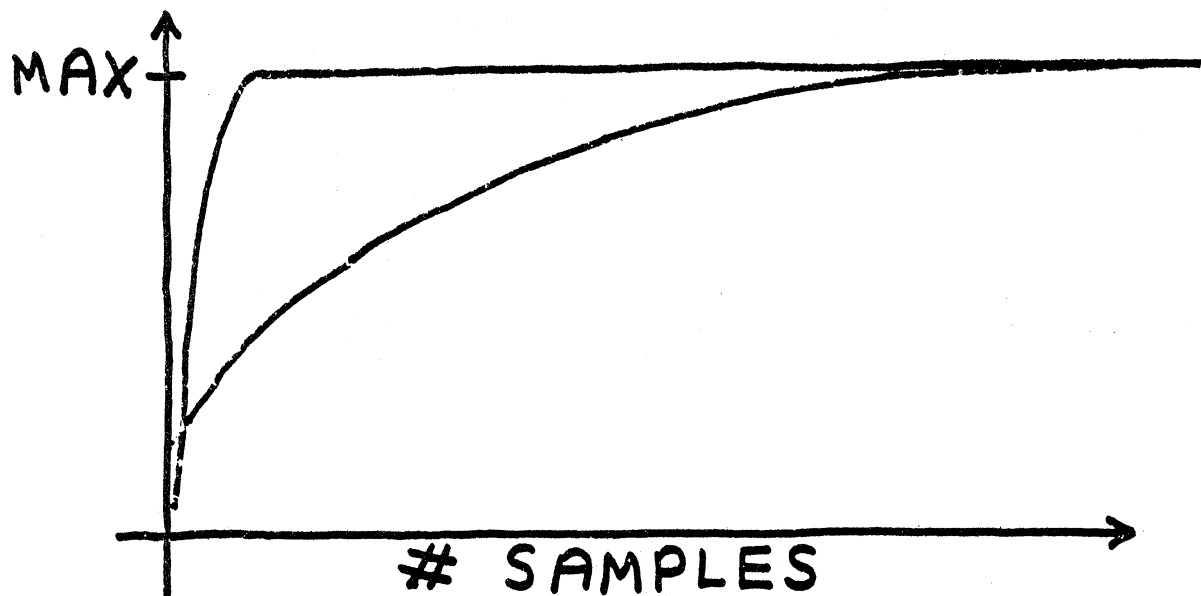
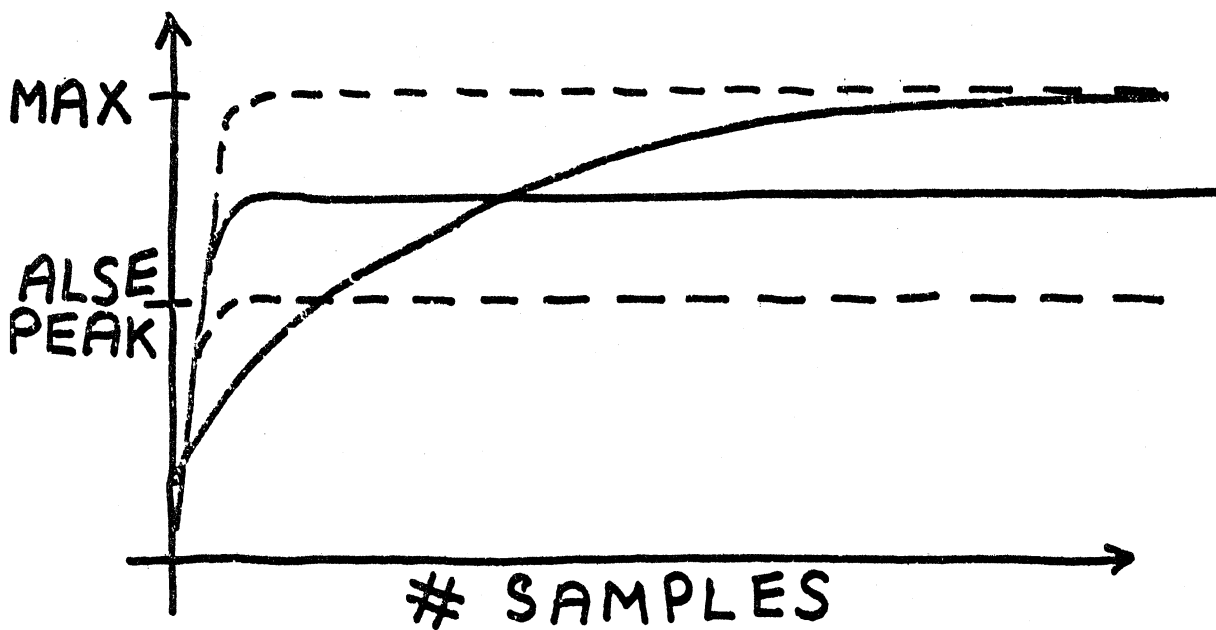


FIGURE 5

PERFORMANCE



UNIMODAL OBJECTIVE



BI-MODAL OBJECTIVE



FIGURE 6

PERFORMANCE

