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

Technical Report

COMPARISON OF GENETIC ALGORITHMS WITH CONJUGATE GRADIENT METHODS

Jack Bosworth
Norman Foo
Bernard P. Zeigler

with assistance from:

National Aeronautics
and Space Administration
Washington, D.C.

administered through:

OFFICE OF RESEARCH ADMINISTRATION ANN ARBOR

February 1972

engn

UNR0354

I. Introduction

A function optimization problem may be defined as follows: Given a real valued function defined on a finite dimensional space, find the points of the space at which the function attains its optimum (minimum or maximum) values. A *direct search algorithm* for solving such an optimization problem is an iterative step-by-step procedure which samples a number of points in the space until a point is found which is apparently optimum.

Function optimization problems requiring direct search algorithms arise from the general area of the design of optimal control systems (Athans and Falb (1966)). The optimal point of view, when applied to the control of aerospace vehicles or chemical processing plants, for example, involves control systems which perform optimally according to some pre-determined criteria of performance. Often the design of such systems leads to function optimization problems which cannot be solved analytically and therefore necessitate direct search algorithms for their solution (Kalman, Falb, Arbib (1969), Lavi and Vogl (1965)).

In many control applications, however, not enough is known about the plant (controlled system) behavior to formulate beforehand a realistic optimal control problem. In this case, one may design a control system from the adaptive control point of view (Bellman (1959), Mishkin and Braun (1961), Feld'baum (1966), Sworder (1966)). An adaptive control system attempts to optimize the performance of the plant "on line", i.e., the controller attempts continually to improve the plant's performance, its actions being based upon its record of past plant responses to control inputs and environmental disturbances. An adaptive controller must possess as essential subcomponents, direct search algorithms which can direct the search toward optimum points of the criterion function (Wilde (1964), Hall and Ratz (1967)).

Thus the successful design of optimal and adaptive control systems rests critically on the existence of useful direct search algorithms for solving function optimization problems. The value of a direct search algorithm in any application depends on its ability in the first place to *converge* (i.e., to actually locate the optimum in a finite time) and secondly, to *converge rapidly* (many algorithms can be guaranteed to eventually locate the optimum but do so much too slowly for practical application). Thirdly, it is important that such an algorithm not be misled by random variations in the criterion function (arising, for example, by digital roundoff error or plant disturbances) into settling on apparent optima far removed from the actual ones.

Genetic algorithms are direct search algorithms which are modelled upon search strategies employed in natural adaptation.

Attempts were made by Fogel, Owens and Walsh (1966) and Bremermann (1966) to implement some of the search strategies employed in natural adaptation. The techniques employed by these workers only superficially resembled those known to exist in nature (Mayr, 1965) and the studies did not yield information concerning the comparative convergence properties or cost and complexity of the genetic algorithms. More sophisticated algorithms employing the mechanisms of crossover, inversion, mutation and reproduction at the genotypic level have been developed by Rosenberg (1967), Bagley (1967), and Cavicchio (1970). These workers obtained experimental results indicating the superiority of the genetic algorithms to competitive methods in the areas of pattern recognition and biochemical adaptation which they explored. Holland (1969a,b,c) has undertaken a systematic theoretical analysis of these methods. His work concerns the existence of an ideal reproductive plan which is "good" in comparison to any other plan, i.e., it sustains only a finite loss over infinite time when compared to any other plan. This criterion is a formalization of the requirements that a search algorithm be "efficient" and "robust" over a broad range of test problems.

Hollstien (1971) developed a class of genetic algorithms for function optimization. He has shown that these algorithms are capable of achieving convergence on functions which are multipeaked and discontinuous where as classical hill climbing methods operate well only on sufficiently smooth single peaked functions.

In this paper we are concerned with the convergence rates of genetic algorithms in comparison with other methods. As a beginning, we investigate the convergence rates of genetic methods relative to those of the conjugate gradient (variable metric) methods (Luenberger (1964), Pearson (1969), Polak (1971)) on test problems typical in the latter area. This is a severe test for the genetic methods since on the one hand they do not employ derivative extraction techniques for guidance (which is available from the analytic structure of the usual test function) and on the other hand the conjugate gradient methods have been honed to the point of extreme efficiency for these functions. Thus from this point of view one may expect relatively inferior performance from the genetic methods. Some positive indications for performance however arise from studies by Rastrigin (1966) and Schumer (1968) which indicate that random step size methods can be more efficient than fixed step size gradient methods. Since Hollstien claims superior performance for his methods over those of

Rastrigin this opens the possibility that genetic methods can compete favorably with the conjugate gradient methods (which are themselves more powerful than the fixed step size gradient methods).

II. Description of Program

As work progressed on our optimization program it naturally underwent a number of modifications. We shall attempt to portray this evolution by describing four stages of development (I,II,III,IV). After the description the theoretical and experimental developments which motivated these modifications will be discussed.

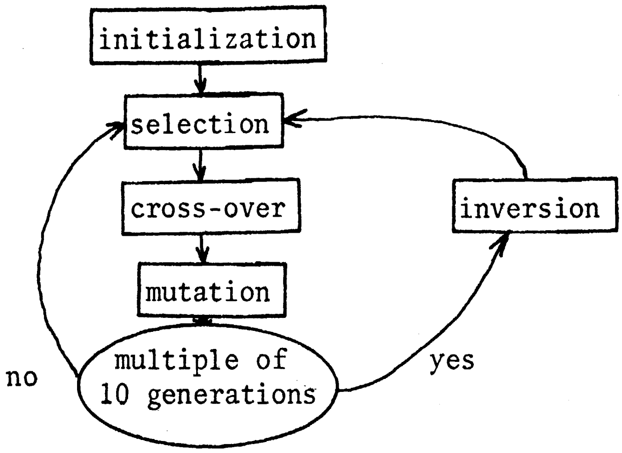
We consider maximization of real valued n-ary functions of the form $f:R^n \rightarrow R$.

A *chromosome* (or *string*) is a list of coordinate values of an n-dimensional vector with an associated inversion pattern. An *inversion pattern* is a permutation of the sequence 1,...,n say $i_1,\ldots,i_n$. If a string is $a_1,\ldots,a_n$ with inversion pattern $i_1,\ldots,i_n$ this means that there is a point in n-space which corresponds to the string such that its i_jth coordinate is a_j. For example, let n=4, and the string be .1, .02, 1.3, -.4 with inversion pattern 1,4,2,3 then the corresponding point is (.1, 1.3, -.4, .02).

The *function value* associated with a string is just the value of the function (currently being optimized) at the corresponding point. Thus the value associated with the above string is f(.1, 1.3, -.4, .02) (*not* f(.1, .02, 1.3, -.4)).

Version I

The basic flow diagram for Version I is as follows:



Forty strings were maintained in four *subpopulations* of ten strings each. Only one inversion pattern was associated with each subpopulation. I.e., any two strings in the same subpopulation had the same associated inversion pattern. A vector called the *utility* vector was maintained giving the function value of each string.

Selection consisted of ordering each subpopulation by function value (i.e., the best string is the one with the highest function value) and then replacing the lowest four strings by the best four strings (in each subpopulation).

Cross-over consisted of picking at random two coordinates for each of the two pairs (7,8) (9,10) of strings in each subpopulation. These are called *pivot* points. Then all coordinate values between and including the pivot points are exchanged between pair members. For example suppose we have a pair of strings $a_1,\ldots,a_5$ and $b_1,\ldots,b_5$ with inversion pattern 1,2,3,4,5 and with pivot points 2 and 4 say. The resulting strings are $a_1b_2b_3b_4a_5$ and $b_1a_2a_3a_4b_5$.

Inversion consisted of ordering the four subpopulations by their best strings,[1] copying the best two subpopulations into the worst two subpopulations, and changing the inversion patterns of the copies as

[1] In each subpopulation the string with the highest function value is found (the best string of the subpopulation) and the subpopulation with the highest "best string" is best, etc.

follows. To change the inversion patterns, two pivot points were chosen for each copy and all strings were inverted about these pivot points, i.e., if $a_1,\ldots,a_5$ is a string of a subpopulation with pivot points 2 and 4 say, then the new string is $a_1a_4a_3a_2a_5$.

Mutation was more complex. A *probability* vector was included in the initial parameter specifications. The vector had four coordinates. Each coordinate specified the probability of using a corresponding method of mutation on any given string.

The methods of mutation were:

1) *Fletcher-Reeves (FR) Mutation.* A version of the Fletcher-Reeves (1960) method which could be applied a controlled number of times q to a point (without reinitialization).[2] When q=1 this reduces to *gradient mutation,* i.e., an approximate gradient was taken at the point specified by the string to be mutated and a "Golden Section" one dimensional search was made along the line from the point specified by the gradient with limits which were initialized.[3]

2) *Uniform random mutation of coordinates.* An integer, the number of coordinates to be mutated, was chosen randomly between 1 and n, say m. m integers (the actual coordinates to be mutated) were chosen randomly between 1 and n, say $i_1,\ldots,i_m$. m numbers (the mutation amounts) were chosen randomly between initialized limits symmetric about 0, say $r_1,\ldots,r_m$. Finally r_j was added to the i_jth coordinate of the point.

3) *Quadratic Gaussian Approximation.* m and $i_1,\ldots,i_m$ were chosen as in 2). 2m numbers were chosen randomly between -1 and 1, say $r_{1,1},r_{1,2},\ldots,r_{m,1},r_{m,2}$. If ℓ is the initialized number determining the "standard deviation" of this mutation, then $r_{j,1}\cdot r_{j,2}\cdot\ell$ is added to the i_jth coordinate of the point for each j=1,...,j=m.

[2] Since everytime the routine is called its remembered gradient is set to 0 this is equivalently a reset mode of operation with reset interval q.

[3] Our Fletcher-Reeves method uses 2n samples for its gradient estimation and 30 samples for its one dimensional search per interation (n is the dimension of the space).

4) *Zero mutation*. The string is left unaltered.

For each of the forty strings one of these four methods of mutation was chosen according to the probability vector and applied to the point corresponding to the string. The resulting point was converted to a string with the same inversion pattern as before and the utility vector was updated by applying the function to the point.

The *initialization* consisted of reading in parameters and initializing the strings to random coordinate values between two bounds, say -2 and 2. The four inversion patterns were all set to 1,2,3,4...n. The utility vector (function value vector) was initialized with the associated function values. All other parameters were considered to be subject to experimental manipulation and initialized accordingly.

Version II

Version I was modified to create Version II in the following ways:

Selection replaced the worst four strings in each subpopulation with four strings from the same subpopulation as follows. The strings are rated 1,...,10 and 7,8,9 and 10 are replaced. String 7 is replaced by string 1. String 8 is replaced by string i where i is (uniform) randomly chosen from 2,3,...,10. String 9 is replaced by string 2 unless i=2 in which case 9 is replaced by 3. String 10 is replaced by a string chosen randomly from those remaining. Thus the best two strings are always duplicated by the selection process. (None of the replacements were made until all strings were chosen.)

Cross-over was done in the same way. Note that the selection now caused cross-over to occur between the best strings and randomly chosen strings, rather than among the best strings themselves.

The four mutation methods of I were used except that 2) was altered as follows:

2') *Cubic Gaussian Approximation.* $r_1,\ldots,r_m$ were chosen randomly between -1 and 1 then added $r_1^3 \cdot \ell$ to the i_jth coordinate.

A fifth method was added:

5) *Uniform Random with Variable Limits.* This method was like the old 2) but the limits between which $r_1,\ldots,r_m$ were chosen were different for different coordinates of the point. Let these limits be $-\ell_1,\ell_1,-\ell_2,\ell_2,\ldots,-\ell_n,\ell_n$. Before this mutation was done the maximum and minimum coordinate values were found for each coordinate, say a_i^* and a_{i*} respectively for the i^{th} coordinate. $\ell_i = a_i^*-a_{i*}$.

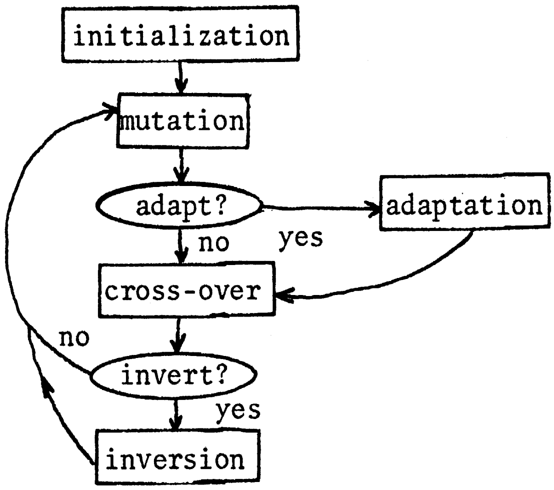
Each string was mutated as before, but when the best string in each subpopulation was mutated (according to the probability vector), the mutant replaced the worst member in the subpopulation (the best string was also saved unmutated).

The major addition to the program structure was a second level "adaptation" routine which controlled some of the parameters previously fixed at initialization. These parameters included the "standard deviation" ℓ used in methods 3) and 2'), the probability vector (determining the disposition toward selecting a particular mutation method). The adaptation was based on a *history vector* which contained information concerning how often and when each mutation was used, the average mutation which resulted in applying mutations 2') and 3) for each subpopulation, and the highest function value present in each subpopulation before mutation.

The adaptation routine used was similar to that of Schumer and Steiglitz (1968). The variance determining parameter ℓ was modified according to whether large mutation amounts or small ones proved more fruitful in producing increases in the function value. A more complete description of this routine is given in Appendix A.

Version III

The flow diagram for Version III is as follows:



The major change introduced was that there was no partitioning of the population into four distinct subpopulations. The population size was determined dynamically but was limited to at most 40. Because separate subpopulations each sharing a common inversion pattern were not maintained some convention had to be adopted in order to achieve crossover between strings with different associated inversion patterns. One possibility, that of allowing crossing over only between strings having the same inversion pattern was rejected (for the difficulties in this strategy see (Bagley (1967)). Instead cross-over was allowed between arbitrary strings with the inversion pattern of the *better* string of the pair determining the alleles to be crossed-over. In essence, the heuristic is that the inversion pattern of the better string is in fact the better inversion pattern. More detail will be given in a moment.

The mutation routine differed from the previous mutation routines in the following ways. A parameter m_1 (determined by initialization) was defined as the number of strings to be mutated. Suppose the program began with m strings (m assumed not less than m_1), then the m_1 strings which had the highest associated function values among the initial m strings were chosen. These m_1 strings were copied. Each of the m_1 copies was mutated using a method chosen randomly with the probability vector determining the frequency of selection of any given mutation method. The mutation methods were the same as 1), 2), 3) and 2') of Version II. Method 5) was not implemented in the Version III mutation routine. (As before, the utility vector was updated and the history vector was maintained.)

The adaptation routine was essentially the same as the adaptation routine of Version II (allowing for the differences in the structure of the history vector). The major difference was that a weighting scheme was introduced to evaluate method effectiveness so that a heavily weighted method had to produce a higher percentage difference in the best function value than a method not weighted so heavily in order to have the ratio of the probabilities of these two methods remain the same. These weights were initialized.

The cross-over routine was altered as follows. Let m_2 be the initialized parameter indicating the number of strings which the routine would operate on. $2 \cdot m_1$ (the number of strings leaving the mutation routine) was assumed greater than or equal to m_2. The best m_2 strings among the $2 \cdot m_1$ strings were chosen. Cross-over initiated by copying the strings present and pairing the copies randomly. Then the alleles (coordinate values) of the string with the higher function value between and including the pivot points were exchanged with the corresponding alleles of the other string. Equivalently the normal cross-over operation is performed except that the inversion pattern of the worse string is replaced by that of the better string before the exchange is begun. After the exchange one of the daughters receives the worse string's inversion pattern (the other daughter inheriting the better string's pattern). For example, if $a_1a_2a_3a_4a_5$ with pattern 12345 and $b_1b_2b_3b_4b_5$ with pattern 54321 are to be crossed over, first create $b_5b_4b_3b_2b_1$ with pattern 12345 and do the cross-over as usual. With pivot points 2 and 4 for example, we obtain $a_1b_4b_3b_2a_5$ and $b_5a_2a_3a_4b_1$. One of these is given pattern 12345 while the other gets 54321.

The number of successive cross-overs was not held at one (as before), but was determined by an initialized maximum bound i subject to the constraint that the process was to be stopped if the population size reached 40. (Note that the population doubles at each successive cross-over and $2^5 = 32$ so $i < 5$.)

The inversion routine always produced m_1 strings. Assuming the entering population size exceeded m_1 the best $\lceil m_1/2 \rceil$ (the least integer greater than $m_1/2$) strings were chosen. Each such string was copied and the inversion pattern of the copy was determined by randomly chosen pivot points as before. (Production was halted when m_1 strings were produced.)

Version IV

Version IV was exactly the same as Version III except that in the mutation routine some of the original m_1 strings were mutated as well. Thus an initialized parameter $m_1' < m_1$ determined that m_1' randomly chosen strings from the original m_1 strings not including the best were to be mutated in the same manner as the m_1 copies already produced.

III. Test Functions

The following functions we used as test functions to be optimized:

1. *Spherical Contours*

$$f_1(x) = \sum_{i=1}^{40} x_i^2$$

2. *Index*

$$f_2(x) = \sum_{i=1}^{40} ix_i^2$$

3. *Index squared*

$$f_3(x) = \sum_{i=1}^{40} i^2 x_i^2$$

4. *Wood*

$$\begin{aligned} f_4(x) &= 100(x_2-x_1^2)^2+(1-x_1)^2 \\ &+ 90(x_4-x_3^2)^2+(1-x_3)^2 \\ &+ 10.1((x_2-1)^2+(x_4-1)^2) \\ &+19.8(x_2-1)(x_4-1) \end{aligned}$$

5. *Valleys*

$$f_5(x) = \sum_{i=1}^{5} i^2(x_{5+i}-x_i)^2+|x_i|$$

6. *Repeated Peaks*

$$f_6(x) = \left(4 \prod_{i=1}^{4} x_i(2-x_i)\right)\left(1+[x_5]-x_5\right)\left(x_5-[x_5]\right)^{[x_5]} \quad \left([x_5]+1\right)$$

$$\text{for } x_i > 0 \quad i = 1,2,3,4 \text{ and } x_5 \geq 1$$

$$= 0 \text{ otherwise}^{4}$$

Functions 1 through 4 are standard in the direct search literature. We invented 5 and 6 to test our hypothezes concerning algorithm behavior.

[4] [x] is the integer part of x, e.g., [1.5] = 1.

NOTE: Functions 1-5 are to be minimized so that in the program f(x) is replaced by -f(x) and the standard maximization formal is satisfied.

IV. Comparison of Genetic and Classical Methods

As stated before, one of our primary objectives was to compare the performance of genetic and classical methods in the realm of numerical function optimization. We hoped to ascertain in this way whether genetic methods could utilize the local structure of analytic functions sufficiently well to compete favorably with classical methods which employ gradient extraction routines.

Our approach was to compare the best of our genetic routines constructed to date with the Fletcher-Reeves method. Of course the results obtained are strictly speaking only relevant to the particular methods compared and the means of comparison. However since the latter were selected with their role of class representatives in mind we have reason to believe that our conclusions may have general validity.

The *experiment* consisted of running Version IV against a control Version II called FR1 in which Fletcher-Reeves is the only mutation method. More specifically, FR1 had the Version II structure except that the mutation routine now had the form: apply the Fletcher-Reeves method with reset interval $q = n$ (where n is the number of variables in the function) to the best string in each subpopulation.

Version IV and FR1 were applied to each of the test functions 2 through 5 with the same initial set of points.

The *results* are shown in Tables 1 and 2. In Table 1, we record for each test function the number of function evaluations taken by FR1 each time the mutation routine is executed. In comparison the number of

TABLE 1

Test Function	Number of FR1 Function Evaluations per generation		Number of Version IV Function evaluations needed to achieve same change in function value	Corresponding change (order of magnitude)
	actual	divided by 4		
2. Index	17,616	4,404	55,800	15
			11,475	1.46
			90	.12
3. Index Squared	17,616	4,404	19,800	1
			4,400	3
			5,500	2
			3,200	1
			3,500	1
			3,500	1
4. Wood	656	164	45	.8
			45	.3
			45	.189
			45	.292
			45	.385
			45	.238
			270	.34
			90	.28
			90	.525
			90	.315
			3780	.583
			3240	.479
			0	.00001
			0	.006
			0	.0
			40,770	1.669
			18,900	.643
5. Valleys	2,100	525	1,566	1.2
			5,220	.71
			1,392	.88
			3,132	1.3

TABLE 2a

Test Function	Function value attained	Number of function evaluations		
		FR1		Version IV
		actual	divided by 4	
1. Spherical Contours	10^{-39}	110*		52,800
2. Index	-1.0×10^{-15}	52,848	13,424	67,725
3. Index Squared	-2.0×10^{-10}	105,696	26,421	40,000
4. Wood	-1.46×10^{-6}	11,152	2,790	68,000
5. Valleys	-3.4×10^{-9}	8,400	2,100	11,310
6. Repeated Peaks	11.999	∞	∞	5,070

*The figure given in the number of function evaluations required by our optimum gradient method (Fletcher-Reeves with q = 1) which converged in one iteration.

TABLE 2b

Test Function	Function value attained	Number of Function evaluations required by Version IV after FR1 hung up
2. Index	1.6×10^{-19}	86,175
3. Index Squared	1×10^{-22}	94,140
4. Wood	1×10^{-14}	200,000
5. Valleys	2.4×10^{-13}	93,544

function evaluations taken by Version IV to achieve the same change in function value is indicated (along with the change in value achieved). The function value attributed to a population is that of its *best* string.

In Table 2a we record the total number of function evaluations taken by the methods to reach the indicated level.

In these tables we have given both the actual number of FR1 function evaluations and this number divided by 4. The latter is a lower bound on the number of function evaluations were the classical Fletcher-Reeves method (i.e., our method in its non FR1 form) to be applied to the best point in the initial population.

It may have become apparent to the reader that we face the difficulty here of comparing the parallel operating genetic methods with the sequential conjugate gradient methods. Our genetic algorithms must start with a number of initial points. The Fletcher-Reeves method begins at one point. We have observed that the rate of convergence of Fletcher-Reeves may be quite variable depending on the nature of the current search region (for example whether it is locally quadratic or near a sharp ridge) and the number of iterations taken since the last re-initialization.

Clearly some kind of aggregate behavior of a method over the search space is required for meaningful comparison. While parallel methods lend themselves more to this form of analysis little is known analytically for either type of method in the present context. Then too which aggregate is to be used: the maximum rate of convergence? the average? the minimum? What if a method fails to converge from some starting points but converges rapidly from others?

As already indicated, our decision was to embed the Fletcher-Reeves method in a Version II genetic program. If we ignore the effects of crossover[5] this is equivalent to applying Fletcher-Reeves to the best point in each of the 4 subpopulations, the number of function evaluations required to reach a given function value level being then four times the number required by Fletcher-Reeves applied to the point which reaches this level first. Knowing before hand which of the four initial points would actually reach this level first we would need only 1/4 of the total. Thus the "divided by four" columns of Tables 1 and 2a represent an "optimistic" estimate of Fletcher-Reeves efficiency. This optimism will be well founded if the variability of convergence is low (so that knowing which starting point is ultimately best is unimportant) and inappropriate if the variability is in fact high in which case the "pessimistic" upper bound is justified.

Our results indicate that except for the behavior on the spherical contours, Wood and Repeated Peaks function there is not a vast difference in convergence rates.

The behavior on the spherical contours functions points out Version IV's lack of gradient extraction facilities. On this function, Fletcher-Reeves (or just optimum gradient) can follow a one-dimension search in the gradient direction directly to the optimum.

The Wood function results indicated that Version IV is not a very good ridge follower. (Its initial progress is comparable to FR1 but it seems to get hung up in mid course though its mutation facilitates enable it to make a recovery).

Repeated Peaks is a multiple peak function and thus should be beyond the abilities on any local hill climbing method. This is substantiated in the fact that FR1 hangs up on the local peak on which it is initiated.

[5] Actually our observations indicate that crossover has little effect in the FR1 context.

Of course for the comparison here to be truly meaningful a global search level should be superposed above the Fletcher-Reeves local search.

The conclusion that convergence rates are comparable on Functions 2 through 5 should be discussed in view of some results of Rastrigin (1963), Schumer and Steiglitz (1968) and Hollstien (1971). The first two references show that on functions of type 1 through 3 a random directional step method can be significantly more efficient than a gradient method with step size fixed at the same value. Hollstien claims superior convergence for his genetic algorithms over the random directional methods.

It follows then that Hollstien's genetic algorithms outperform the gradient methods with fixed step size.

It is crucial here to note that the gradient methods referred to by Rastrigin are of the fixed step size type and not of the conjugate gradient class which we considered. The efficiency of the latter conjugate gradient methods (of which Fletcher-Reeves is a good representative[6]) is known to a exceed that of the fixed step gradient.[7] Thus the question remains open as to how the random direction methods compare with the conjugate gradient methods and hence how Hollstien's genetic methods compare with the gradient methods. Our present results thus add essentially new information to this comparison.

[6] In a comparison of 7 conjugate gradient methods including the well-known ones, Pearson's (1969) results show that in terms of the number of one-dimensional searches, Fletcher-Reeves is superior to all others (except Newton Raphson) when operated in the reset mode (as it is here) on the Rosenbrock and Wood functions. Thus we chose Fletcher-Reeves since it is both more simple and efficient on the "well-behaved" functions we considered. (On the "penalty functions" considered by Pearson the situation is drastically reversed with Pearson's method #3 coming out well on top.)

[7] This can be seen in any of the texts referred to in the literature survey and is essentially due to the use of one-dimensional rather than the much more costly n-dimensional searches.

Actually, Schumer compares his method with Newton Raphson on $\sum_{i=1}^{n} x_i^2$ (our function 1) and $\sum_{i=1}^{n} x_i^4$ and finds it inferior on the former for $n < 78$ and superior on the latter for $n > 2$. The comparison is in terms of the number of function evaluations and it appears that Schumer's method increases linearly while Newton Raphson increases quadratically in this regard (essentially because second partial derivatives must be estimated). As we have indicated the function evaluations per iteration required by Fletcher-Reeves increase only linearly in dimension and on the $\sum_{i=1}^{n} x_i^2$ and $\sum_{i=1}^{n} x_i^4$ functions it should far surpass Newton Raphson in this measure.

In fact, Table 2a shows that our Fletcher-Reeves requires only 110 samples compared to the 330 required by Schumer's method and the 1500 required by Newton Raphson[1] (Data taken from Schumer's Figure 4). Thus the classical Fletcher-Reeves should be uniformly better than Schumer's method on Spherical Contours for any finite dimension.

It is interesting also that Schumer's method proved not very effective as a ridge follower as indicated by its inferior performance on Rosenbrock's function.

It should be noted that Version IV was able to reach much lower function value levels than was FR1. This is shown in Table 2b which gives Version IV's behavior starting from the levels indicated in Table 2a. The latter levels are those for which FR1's progress terminated. (This may be an artifact of our Fletcher-Reeves realization.)

[1] Actually the difference is even more striking when it is considered that Fletcher-Reeves reached 10^{-39} from point (2,2,...,2), while Schumer's data are for the level 10^{-8} starting from (1,1,...,1). Note that Version IV's performance fell in between the Schumer and Newton Raphson methods.

V. Evolution from Version I to Version IV

5.1 Mutation and Second Level Adaptation Routines

Initially, only mutation method 1) was used. It involved a uniform random selection of coordinates (loci) and values (alleles). Methods 3), 2') and 5) were introduced in order to bias the distribution toward small changes. This improved convergence by helping the system move off false resolution ridges (Wilde, 1965). Later we discovered that adding a second level routine (Version II) to modify the biasing on the basis of past experience considerably improved performance. Table 3a indicates the effectiveness of the adaptation routine. Our analysis of the reasons for the improvement obtained is as follows.

As a run progresses the best alleles must be changed less in order to improve. For this reason, the standard deviation of a random mutation must decrease in order to improve the probability of a better mutation. To this end we implemented some history vectors and added a program to adapt the mutation parameters in a Bayesian approximation. Thus, if a smaller mutation had worked best, the standard deviation was decreased and like-wise for a larger mutation, If there has been no improvement in function value over the period of history the standard deviation was halved assuming that it had been too large. When the parameters became too small for the accuracy of the machine, they were reset to maximal values.

It was apparent that the kind of mutation which worked best at one point in a run was sometimes different for a different part of the run. For this reason more history was kept and the probabilities of the different mutation methods were changed. This seems to work but does not usually give marked improvement in the performance of the system.

Non-uniform distributions worked better than uniform ones when no adaptation was applied since there was a higher probability of small change.

With the adaptation, uniform works best under certain conditions since the probability of making the right size change is higher and adaptation can progress faster. Under different conditions the uniform is more likely to put the adaptation parameter in a "quasi-stable" state where change is quite smooth but too slow to be useful.

By a "quasi-stable" state we mean a situation in which the adjusted parameter is maintained for a long period of time at a suboptimal value. This can happen in our present system since we include no random or regular reset. Resetting of the parameter occurs only when it has passed below a preset limit. Thus a situation in which the parameter is not below the preset level but is still too small to cause significant changes in the function values of mutated points will result in "quasi-stable" state since the information fed back to the adaptive routine is insufficient to cause a directed change in the parameter setting.

We tested the more complicated variable limit mutation (5) against the simpler quadratic mutation (2). Table 3b indicates that 5) was indeed better than 2) on the two functions shown. However, used with the additional adaptation routine the extra variability of 5) was redundant in view of the adaptive flexibility (i.e., controllable variability) introduced by the latter (adaptation) routine.

We were interested in the extent to which gradient information could be encorporated into the genetic algorithm structure. Employing Fletcher-Reeves as a mutation routine does not give much of an answer to this question since on the test functions employed it tends to speed up convergence to such an extent that the essential genetic elements (cross-over

TABLE 3 a

Number of function evaluations required by Version I v.s. Version II. (All parameters are set to the same values except that Verstion II employs a second level adaptation routine).

Function	Value Attained	Version I	Version II
1. Spherical contours	-2.045E+1 *	90	90
	-5.28	900	1050
	-3.78	1350	1175
	-2.97	2250	1350
	-2.48	2700	1440
	-1.80	4400	1440
	-1.36	5500	1525
	-1.26	5840	1620
	-1.00	10,750	1700
	- .753	13,400	1890
	- .472	14,400	2150
	- .268	17,820	2700
	- .218	28,600	2790
	- .217	>38,300	2790

*aEb is Fortran for a x 10^b

TABLE 3b

The number of generations required by Version II using quadratic mutation (2) v.s. variable limit mutation (5).

Function	Value Attained	(2)	(5)
2. Index	-700	10	20
	-400	15	50
	-300	36	70
	-200	75	110
	-100	190	170
	- 80	260	190
	- 60	310	220
	- 40	550	260
	- 20	>4200	470
4. Wood	- 15.0	7	10
	- 10.0	9	
	- 9.0	10	20
	- 4.0	12	40
	- 2.0	15	60
	- 1.0	46	70
	- .5	60	80
	.1	>700	170
	.01	>700	370

TABLE 3c

The number of generations required by Version II using a pure gradient mutation (1) v.s. a mixed strategy using 1) with probability 3/4 and quadratic random (3) with probability 1/4.*

Function	Value Attained	1)	3/4(1)+1/4(3)
3. Index squared	-100	9	10
	- 10	34	34
	- 8	43	35
	- 6	55	41
	- 5	64	45

*Note that 1) also uses more function evaluations per generation than does 3/4(1)+1/4(3).

TABLE 3d

The number of generations required by version I with "best saved" strategy versus "best not saved."

Function	Value Attained	Best Saved	Best not Saved
3. Index Squared	-.198E5	10	10
	-.11E5	20	30
	-.7E4	40	50
	-.5E4	50	90
	-.4E4	70	120
	-.3E4	90	160
	-.2E4	110	310
	-.1E4	190	> 4700

and inversion) do not play much of a role. However when we used gradient mutation (i.e., Fletcher-Reeves with q = 1) we found that it worked better in conjunction with random mutation than alone (see Table 3c).

It was apparent that the mutation often changed the best string in a given population for the worse. When we saved the best string in each subpopulation by replacing the worst string with the mutation results from the best string, the performance was increased several fold. (Table 3d).

5.2 Inversion and Crossover

We called the Version I kind of cross-over *best-with-best* because cross-over only occurred between the best strings in each subpopulation. Since mutation can cause good alleles (coordinate values) to appear in a string and still make the string bad by making some alleles bad, best with best cross-over does not use all its potential. For this intuitive reason and other reasons based on our theoretical concept of crossover we tried crossing over the best strings with randomly selected strings so that only part of the time are the best strings crossed over with each other (see the description of Version II). This improved performance considerably as shown in Table 4.

We examined the effectiveness of crossover and inversion where no mutation routines were used. Here one expects that the ultimate function value level attained is governed by the alleles present in the initial population (since none can be introduced by the mutation) so the real test is whether crossover and inversion can operate to select the best alleles of those available. That this is possible is indicated in Table 5, where the alleles in the final population are no worse than the second best of those initially available.

We also tested the effectiveness of crossover in bringing together

TABLE 4

The number of generations required to reach indicated level by crossing over best with best (BB) v.s. best with random (BR).*

Function	Function value attained	BB	BR
1. Spherical contours	-500	10	6
	-400	19	15
	-300	48	36
	-200	190	75
	-100	>400	190
3.. Index squared	-10,000	16	16
	- 8,000	30	22
	- 6,000	50	34
	- 4,000	108	75
	- 2,000	>2700	200
4. Wood function	-15	2	7
	-10	9	9
	- 9	21	10
	- 4	32	12
	- 2	>600	15

*Version I using mutation 2) (uniform random)

TABLE 5

The effectiveness of Verison IV with no mutation and 1 crossover per generation.

Function	The two smallest* values of alleles in 1st four co-ordinates available in initial population				After generation 12 only one string remained:			
2. Index	1	2	3	4	1	2	3	4
	.3835	.0488	.0765	.1181	-.6048	.0488	.1774	.1181
	-.6048	.0976	.1774	.2021				

*Clearly, for the index function, the smallest are the best.

"good" alleles in another way. Table 6 shows that Version IV without a crossover routine was unable to achieve the ultimate performance of Version IV using 2 crossovers per generation.

The effectiveness of inversion was similarly tested (Table 7).

5.3 Comparison of Versions II and III

The motivation for constructing the version III system was as follows:

It seemed probable that a lot of "excess baggage"[8] was being carried along in the four subpopulations with the result that more function evaluations were being used than was necessary. However, were we to reduce the *number* of subpopulations to two or three, only two inversion patterns would be compared in general. Thus on a function some of whose variables are linked, inversion patterns would not be tested rapidly enough to improve the effectiveness of cross-over. On the other hand, if the *size* of the subpopulations were reduced to say five or six strings each, crossover would have been less effect. Thus it appeared that one must do without subpopulations to achieve the smaller sized (total) population.

But subpopulations were maintained primarily to preserve a single inversion pattern. Comparing of subpopulations was thus a test of the effectiveness of inversion patterns. How can this be achieved without subpopulations?

Doing away with subpopulations also forces the question: What strings will be crossed over and how? Suppose that only strings having the same inversion pattern may be crossed over. Suppose the function has n variables. Then there are n!/2 essentially different inversion patterns since any permutation of the variables is an inversion pattern, but turning any inversion pattern end for end preserves clumpings. This means that for functions with more than three or four variables cross-over

[8] i.e., redundant information

TABLE 6

Number of function evaluations required by Version IV with and without crossover (all other parameters fixed).

Function	Value Attained	Without Crossover	With Crossover*
2. Index	-6.03E2	75	225
	-3.4E2	750	450
	-1.65E2	1500	1125
	-6.87E1	2250	2475
	-2.05E1	3000	4500
	-8.07E0	4500	6750
	-3.19E0	6000	8100
	-.9.99E1	7500	10,080
	-5.82E-1	9000	10,800
	-3.59E-1	10,500	11,210
	-2.68E-1	12,000	11,700
	-1.80E-1	15,000	12,150
	-7.7E-2	22,500	13,300
	-4.9E-2	30,000	13,300

*using 2 consecutive crossovers

TABLE 7

Number of generations required by Version I with and without inversion (all other parameters fixed).

Function	Value Attained	No Inversion	Inversion
5. Valleys	6.7	10	10
	4.5	30	30
	4.0	40	30
	3.0	60	50
	2.2	90	60
	2.0	150	70
	1.5	170	100
	1.3	260	140
	1.0	380	150
	.9	440	340
	.7	520	360
	.6	570	360

would take place very seldom in a set of strings which have individual inversion patterns. Therefore a more general kind of cross-over must be employed. As already indicated, we tried crossing over two strings with different inversion patterns by picking two pivot points as before but applying the pivot points to the string with the better function value and simply exchanging the alleles involved with the corresponding alleles of the worse string no matter where those alleles are in the string.

This kind of cross-over allows unrestricted cross-over with only a slight computing cost to find the "corresponding alleles". *It assumes that the string with the better function value usually has the better inversion pattern.* That is, that the inversion pattern clumps the right variables. Although this type of cross-over is only slightly different from the first, its consequences are more difficult to predict. It seems to be about half as effective at finding the best inversion pattern.

The results obtained from the version III and IV systems are often uncertain because they have more than a dozen parameters. The purpose of having open so many parameters was that we wished to be able to test hypotheses which we had formulated as a result of our experience with the version II system. These parameters have proved to be quite interdependent. That is, we find that for any reasonable setting of all but one parameters, (that one being arbitrary), varying the single parameter has a strong effect on the efficiency of the system. However, having found the optimal value for that parameter with the others fixed, changing some of the other parameters frequently changes the optimal value for the one parameter by a large amount. An important project for the future will be to chart the interrelations involved.

However we were able to show that there were settings of Version III parameters which yielded performance much superior to Version II (Table 8)

TABLE 8

The number of function evaluations required to reach indicated function value by Version II v.s. Version III.

Function	Value Attained	Version II	Version III
3. Index Squared	8000	630	180
	4700	1,260	440
	3200	2,520	630
	2200	2,835	1,080
	950	4,400	1,710
	700	5,350	1,800
	500	7,550	2,250
	200	13,200	3,150
	80	18,300	3,780
	40	21,400	4,590
	10	34,300	6,930
	7	35,900	9,000

thus justifying the change in system structure.

VI. Conclusions

If the reader finds himself unable to formulate a clear statement of the results of our work to date, let us assure you that we feel ourselves to be in the same position. We have constructed a class of algorithms which are sufficiently complex to be highly interesting, but which at the same time are not readily amenable to analytical study and classification.[9] Thus, without the benefit of theoretical guidance we are reduced to stab in the dark experimentation. Moreover, since a single optimization run takes hours to complete our rate of progress in obtaining data is not as fast as our enthusiasm demands.

Within these constraints however, we have obtained suggestive results concerning the comparative behavior of genetic and conjugate gradient algorithms, and we have also come to some conclusions concerning the effectiveness of various subcomponents of the genetic algorithms. We have for example demonstrated optimization problems where encorporating individually the crossover and inversion operators actually does achieve a significant improvement in the rate of convergence (Tables 5,6,7). Clearly much remains to be done in confirming or disconfirming these conclusions.

[9] We look forward to a forthcoming book by J.H. Holland on adaptive systems for possible help in this direction. Also some preliminary analysis will appear in our Report (Foo and Bosworth, 1972).

APPENDIX

The following is a more mathematical description of the adaptation used.

Let f_i denote the function value of the best string (the one with the highest function value) just before the i^{th} mutation following the last adaptation. Let f' denote the function value of the best string just before the mutation which occurred just before the last adaptation. In the case of the version I system i has values between 1 and ten. i has values between one and an initialized integer, say i_0, in the case of the version II system. Let i_0 denote i_0 for version II and ten for version I. Let $d_i = \frac{f_i - f_{i-1}}{f_{i-1}}$ for $i=2,\ldots,i_0$ and $d_1 = \frac{f_1 - f'}{f'}$. d_i is the percentage difference in best function value between generations i-1 and i. Let $w_i = d_i(i+1)/2$.

In the case of subpopulations, each has its own best string and its own average mutation for each generation. This requires double subscripting. To avoid this we will only consider the version II system. The same methods are used and an exact understanding of the version I system may be gained from the program listings. Let a_i denote the average of all mutations used in the i^{th} generation following the last adaptation. Let a' correspond to f'.[10] The theoretical average of the a_i over an infinite number of trials

[10] When a string is mutated using one of the methods which use the adaptation parameter ℓ, a random number of coordinates are chosen to be mutated, say i ($1 \le i \le n$). If the "cubic" mutation is used an r_j is chosen for the j^{th} coordinate if it is to be mutated where $r_j \in [-1,1]$. The absolute values, $|r_j|$, of these i numbers are averaged, the average being say b_k where the string mutated was the k^{th} string to be mutated that generation. If the "quadratic" mutation is used an r_{1j} and an r_{2j} are chosen for the j^{th} coordinate if it is to be mutated where $r_{mj} \in [-1,1]$. The $2 \cdot i$ numbers $|r_{1j}|$, $|r_{2j}|$ are averaged, the average being b_k for the k^{th} string mutated that generation. Let a_i denote the average of all the b_k in the i^{th} generation.

is .5 since the r_i and r_{ji} have absolute values uniformly distributed between 0 and 1. Let a^* and a_* denote respectively the maximum and minimum of the a_i's including a' excluding a_{i_0}.

$$\text{Let}\quad \ell' = \left(\left(\left[\frac{\left(\sum_{i=1}^{i_0-1} w_{i+1}\cdot a_i\right)+w_1\cdot a'}{\sum_{i=1}^{i_0} w_i}\right]-.5\right)\cdot\frac{\ell}{2\cdot a^*-a_*)}\right)+\ell\ .$$

If ℓ' is greater than $\frac{3}{2}\cdot\ell$, ℓ is replaced by $\frac{3}{2}\cdot\ell$. If ℓ' is less than $\frac{1}{2}\cdot\ell$, ℓ is replaced by $\frac{1}{2}\cdot\ell$. If $\frac{1}{2}\ell \le \ell' \le \frac{3}{2}\ell$, ℓ is replaced by ℓ'. If the new ℓ was less than an initialized constant, ℓ was reset to the value ℓ was initialized to. The $\frac{1}{2}$ limit in the change of ℓ is arbitrary obviously. The above is a Bayesian approximation based on the assumption that the amount of usable information stored in the history vector pertaining to the effect of a generation of mutation on generations following the next one is negligible.

This same assumption is made when adapting the probability vector. Let p be the probability of the mutation method under consideration (found from the probability vector). If p is 0 the method was not used so go to the next method. Let k_i be the number of strings to which the method was applied correspond to the a_i (from adapting ℓ). Let k' be the number just before the last adaptation (like a' and f'). If m strings were mutated each generation, over an infinite number of trials the k_i should average $p\cdot m$. Thus let

$$p' = \left(\left(\frac{\left[\left(\sum_{i=1}^{i_0-1} w_{i+1}\cdot k_i\right) + w_1\cdot a'\right]}{\sum_{i=1}^{i_0} w_i} - p\cdot m\right)\cdot p/10\right) + p$$

p could be changed by no more than one tenth in the same manner as ℓ could be changed by no more than one half. Therefore p was normally set to p'. The probabilities in the vector had upper and lower limits as to numerical size. The "probabilities" in the probability vector were not normalized. You can easily see that this has no effect on the above computations since the p used there is a true probability derived from the probability vector. The limits on the vector were programmed so that no value in the probability vector could be greater than 20.0 or less than 0.5 or 0.1 in the version I or version II systems respectively.

VII. References

Athans, M. and Falb, P.L. (1966) "Optimal Control: An Introduction to the Theory and Its Applications" McGraw-Hill.

Bagley, J.D. (1967) "The Behavior of Adaptive Systems Which Employ Genetic and Correlation Algorithms" Doctoral Thesis. Department of Computer and Communication Sciences, The University of Michigan.

Bellman, R. (1959) *Adaptive Control Processes: A Guided Tour*, Princeton University Press.

Bremermann, H.J.; Rogson, M.; Salaff, S. (1966) "Global Properties of Evolution Processes" in *Natural Automata and Useful Simulations*. Spartan Books.

Brioschi, F. and Locatelli, A.F. (1967) "Extremization of Constrained Multivariable Function: Structural Programming" *IEEE Trans. on Sys. Sci. and Cy. SSC-3*, 2.

Cavicchio, D.J. (1970) "Adaptive Search Using Simulated Evolution" Doctoral Thesis. Department of Computer and Communication Sciences, The University of Michigan.

Cohen, A.I. (1971) "Rate of Convergence of Several Conjugate Gradient Algorithms" *Fifth Annual Princeton Conference on Information Science and Systems*.

Cragg, E.E. and Levy, A.V. (1969) "Study on a Supermemory Gradient Method for the Minimization of Functions" *Journal of Optimization: Theory and Application, 4*, 3.

Davidon, W.C. (1966) *Variable Metric Method for Minimization*. Argonne Nat. Lab. ANL-5990 (Rev. 2).

Feld'baum, A.A. (1966) *Optimal Control Systems*. Academic Press.

Fletcher, R. and Reeves, C.M. (1964) "Function Minimization by Conjugate Gradients" *The Computer J., 7*, pp. 149-154.

Fletcher, R. and Powell, M.J.D. (1963) "A Rapidly Convenient Descent Method for Minimization" *The Computer J., 6*, pp. 163-168.

Flood, M.M. and Leon, A. (1963) "A Direct Search Code for the Estimation of Parameters in Stochastic Learning Models" Mental Health Research Institute, The University of Michigan, Preprint 109.

Fogel, L.J.; Owens, A.J.; Walsh, M.J. (1966) *Artificial Intelligence Through Simulated Evolution*. John Wiley and Sons, Inc.

Foo, N. and Bosworth, J.L. (1972 in preparation) Algebraic, Geometric, and Stochastic Aspects of Genetic Operators.

Hall, C.D. and Ratz, H.C. (1967) "The Automatic Design of Fractional Factorial Experiments for Adaptive Process Optimization" *Information and Control, 11,* pp. 505-527.

Hartmanis, J. and Stearns, R.E. (1969) "Computational Complexity" *Information Sciences.*

Hill, J.D. (1969) "A Search Technique for Multimodal Surfaces" *IEEE Trans. on Sys. Sci. and Cy., SSC-3,* January.

Holland, J.H. (1969) "A New Kind of Turnpike Theorem" *Bulletin of the American Math. Soc., 75,* 6.

Hollstien, R.B. (1971) "Artificial Genetic Adaptation in Computer Control Systems" Doctoral Thesis. Department of Computer Information and Control Engineering, The University of Michigan.

Kalman, R.E.; Falb, P.L.; Arbib, M.A. (1969) *Topics in Mathematical System Theory.* McGraw-Hill Book Co.

Lasdon, L.S. (1971) *Optimization Theory for Large Systems.* MacMillan.

Lawler, E. (forthcoming book) *Combinatorial Optimization Theory.*

Leon, A. (1965a) "A Classified Bibliography on Optimization" In *Recent Advances in Optimization Techniques.* Lavi, A. and Vogl, T.P., eds. John Wiley and Sons.

___, (1965b) "A Comparison Among Eight Known Optimization Procedures" In *Recent Advances in Optimization Techniques.* Lavi, A. and Vogl, T.P., eds. John Wiley and Sons.

Luenberger, D.G. (1969) *Optimization by Vector Space Methods.* John Wiley and Sons.

Mayr, E. (1965) *Animal Species and Evolution.* Harvard University Press, Cambridge.

Miele, A. and Cantrell, J.W. (1969) "Study on a Memory Gradient Method for The Minimization of Functions" *Journal of Optimization Theory and Application, 3,* 6.

Mishkin, E. and Braun, L. (1961) *Adaptive Control Systems.* McGraw-Hill Book Co.

Pearson, J.D. (1968) "Decomposition in Multivariable Systems" *IEEE Trans. on Sys. Sci. and Cy., 1. SSC-4.*

___, (1969) "Variable Metric Methods" *The Computer Journal, 12,* 2.

Polak, E. (1971) *Computational Methods in Optimization - A Unified Approach.* Academic Press.

Rastrigin, L.A. (1963) "The Convergence of the Random Search Method in the Extremal Control of a Many Parameter System" *Automaton and Remote Control, 24,* pp. 1337-1342.

Rosenberg, R. (1967) "Simulation of Genetic Populations with Biochemical Properties" Doctoral Thesis, Department of Computer and Communication Sciences, The University of Michigan.

Rosenbrock, H.H. (1960) "Automatic Method for Finding the Greatest or Least Value of a Function" *The Computer Journal, 3,* pp. 175-184.

Schumer, M.A. and Steiglitz, K. (1968) "Adaptive Step Size Random Search" *IEEE Trans. on Aut. Control, AC-13,* 3.

Shekel, J. (1971) "Test Functions for Multimodal Search Techniques" *Fifth Annual Princeton Conference on Information Science and Systems.*

Spang, H.A. (1962) "A Review of Minimization Techniques for Non-Linear Functions" *SIAM Review, 4,* pp. 343-365.

Sworder, D. (1966) *Optimal Adaptive Control Systems.* Academic Press.

Wilde, D.J. (1964) *Optimum Seeking Methods.* Prentice-Hall.

Wood, C.F. (1964) "Review of Design Optimization Techniques" Westinghouse Research Laboratories, Science Paper 64-SC4-361-P1.

Zeigler, B.P. (1969a) "On the Feedback Complexity of Automata" Doctoral Thesis, Department of Computer and Communication Sciences, The University of Michigan.

___, (1969b) "On the Feedback Complexity of Automata" *Proceedings of The Third Annual Princeton Conference on Information Science and Systems.*