1 Introduction

Computer Genesis

And God completed on the seventh day
the things He had done;
and in the seventh day He rested or *ceased*
of all the things that He had finished

Genesis [II:2]

... The End and the Beginning are only dreams!
without being born, without dying and without changing
the spirit always remains...

Bhagavad-Gita
(The Song of the Lord)
Mahabarata

Ever since Darwin set forth his theory on natural evolution and the survival of the fittest there has accumulated an impressive amount of experimental evidence to support it. This theory, so natural to our generation, faced seemingly unsurmountable arguments against it (or so it would seem to a typical 19th century occidental). For, how could hummingbirds and alligators, for instance, be said to share a common ancestor? However, there is no *need* for a master plan to explain the complexity of Nature. There is no *need* for a superior Will to mold the destiny of Man.[1] Things are vastly simpler than our forefathers would believe and, at the same time, amazingly complex. Changes occur not in leaps, but in minute steps. And it is the yuxtaposition of millions upon millions of such minute steps which, after the fact, gives rise to a seemingly purposeful design.

For years a battle raged between evolutionists and creationists. Then, by mid-twentieth century DNA and the genetic code were discovered and understood. The chemical workings of the genetic code were slowly deciphered and it turned out, in the end, that the double helix of DNA was Nature's answer to an efficient way to encode, in a quaternary alphabet, the delicate workings of a living being. Furthermore, it turned out that, regardless of how humble the living being, the genetic code is practically always

[1] For a lucid defense of this point of view see [DAWK87].

the same.[2] From the lowliest of virus (including the dreaded HIV) to the human genome the code is based on the same alphabet; that when this alphabet is interpreted as instructions every instruction is encoded in the same way. And this fact remains, in my opinion, the strongest argument in favor of a common origin to all living things upon Earth.

Now, whether we accept evolutionism or not, it would seem to be a rather effective process to learn about. If indeed it was able to give rise to such a staggering variety of lifeforms, as evolutionists would have us believe, it would be rather interesting to fathom its innards. Simply look at the biodiversity and one cannot but help to admire the exquisite complexity and delicacy implicit in even the most simple living organisms. It would certainly be an asset to find out how is it that, through this evolutionary process, nature has learned all it knows: the temperature that our bodies must maintain and how to keep it, the way to extract oxygen dissolved in water, how to design bodies which can fly, the best way to interconnect a vast amount of neurons to give rise to the epiphenomenon of intelligence, how to manufacture a living being from two others. This list is enormous: as large as living variety.

If we attempted to capture the essence of the success of evolution we would, perhaps, come up with a recipe such as the following:

a) Store the possible solution to a given problem (i.e. "Do I need wings to fly?") in the individual itself (that is, give wings to the individual).
b) Test the goodness of the individual in question in front of the environment (that is, "Once given wings, can I really fly?")
c) Allow the survival of those individuals which are best fit with preference over other individuals (That is, when flying for fish, a pelican is probably better at it than a pterodactyl would have been. Hence, feathered primitive flying beings had a larger share of fish that non-feathered ones. Hence the former had more surviving descendants than the latter. Hence, after some time, there were more feathered flying living beings than non-feathered flying living beings).
d) From time to time, mix the solution stored in the surviving individuals to get new mixtures of attributes and, possibly, better solutions to the problem in hand (that is, let a bird with hollow bones breed with a bird whose bones are not[3] and, therefore, test which of the two is a better alternative).
e) Periodically enhance the set of possible solutions by disrupting the stored information (That is, alter the code at random. If this is done once in a while chances are that, once in a while,[4] the code resulting from the said alterations will, in fact, give rise to enhancements of the proposed solution).

---

[2] A notable exception is the case of some viruses.

[3] This is an unrealistic example. Hollow bones were inherited by birds from reptiles. But for the purpose of this example we shall allow it.

[4] This process of mutating the genetic code is not something that Nature "chooses" to do. It is a result of errors in the reproduction process, on the one hand, and of external agents on the other. For instance, cosmic rays and chemical alterations do have an effect on the efficient replication of the encoded information.

Repeat steps (a) to (e) millions of times and a bacteria will possibly give rise to a multi-celled organism; an arthropod will possibly give rise to a fish; a fish will give rise to a bird, and so on.[5]

The question is, can we copy such a scheme?

Clearly we do not have the millions of years that Nature has invested to find the long neck of a giraffe as the solution to the problem of reaching the leaves of acacia. But there seems to be a way out of this problem, if only we could change the time scale; if only we could cope with the genetic encoding problem; if only we could reproduce the pressure of the environment on the individuals; if only we could simulate the apparent randomness of Nature. If we could do these things we could go one step further and test alternatives which Nature has not tried yet.

It turns out that it is possible to take advantage of such a proven method without trying to wait for inordinately long periods of time because we may use a computer as a vehicle to simulate (if only partially) a controlled environment. And it is upon this idea that Genetic Algorithms have been conceived and have grown.

1.1 What is a Genetic Algorithm?

Genetic Algorithms (GAs) are computer-oriented procedures which attempt to characterize the essentials of a system by partially simulating a process of natural selection.

We say that they are computer-oriented because, as stressed in the introduction, natural selection occurs over large ensembles of individuals in relatively long periods of time. Since we expect to achieve the said characterization in short periods of time (typically from a few seconds to, maybe, a few hours) it is obvious that the partial simulation of natural selection mentioned above has to take place in simulated time. This simulated time is dependent on the clock rate of the host computer but usually telescopes real time units into millions of simulated time units. That is, we expect to simulate millions of events in a few seconds.

Thence lies the power of GAs: by simulating a large number of simple events we expect to achieve results which are statistically significant. In this sense GAs are the successors of Monte Carlo simulation. Unlike Monte Carlo simulation, however, a GA is a self-modifying simulation where its behavior is a function of how well the method performs and whose long term operation is adaptive. That is, the GA learns from experience and moves toward a better option as the simulation proceeds, i.e. it evolves.

Of course, the extent to which we may affirm that a system is being characterized depends largely on the model adopted. Much of the art of Evolutionary Computation in general and GAs in particular depends on one's ability to reflect in the model the true

---

[5] We do not mean to imply that the described transitions are literal. As already mentioned, the changes are minute and imperceptible. It is the accumulation of these minute evolutionary changes that gives rise to an apparent sudden creation.

nature of the system. This is true of all kinds of models be they mathematical or otherwise. Where the GA differs sharply from other methodologies is in the relative uniformity achievable in the process of modeling.

There is, perhaps, no simpler way to explore the details of what a GA is and what it can do than to present an example. In what follows, therefore, we have chosen a rather simplified but quite practical problem.

1.1.1 Steps of the Genetic Process

Let us assume that we want to find the largest value of the function $Y = 2X^2 - 3$ subject to the conditions: a) X and Y may only take whole (integer) values; b) X must be positive and c) X < 16.

This problem would hardly deserve more than a cursory examination to find that $\max(2X^2 - 3) = 447$ (when X = 15) given the above conditions. The corresponding graph is shown in figure F.1.1.1.1. However, this rather simple example will allow us to identify the main components of the genetic process.

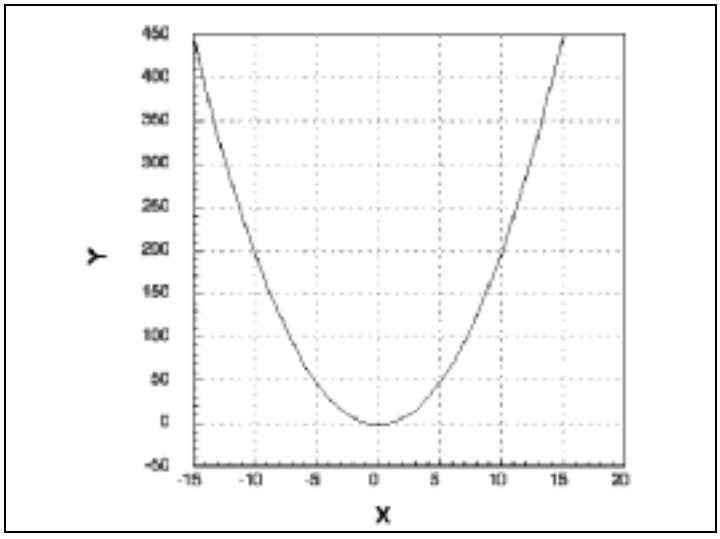

Figure F.1.1.1.1. $Y = 2X^2 - 3$

1.1.1.1 Encoding

To begin with, we shall encode our hypothetical solution in a binary string. This rather direct choice is not the only one. We shall choose to encode the candidate solution in 4

bits. This four bits shall stand for a numerical value ranging from 0 to 15. Although, as we shall see, other encoding schemes are indeed possible, we will accept this weighted binary encoding for our example. The purpose of this choice is twofold: first, we suspect that the solution we are seeking may be represented by, at most, 4 bits and, thus, there is no need for a larger string; second, it allows our example to remain simple enough.

1.1.1.2 Initial Population

We assume that we know nothing about the characteristics of the solution to our problem. Hence, we shall let the computer pick for us the first N candidate solutions, where we arbitrarily let N=10. Once having done so, we find that the initial population is the one shown in Table T.1.1.1.1.

Notice that here we have implicitly described, for the first time, what we mean by "population". Tacitly we have also defined "individual". By *individual*, therefore, we mean a binary encoding of a possible solution. By *population* we mean a set of individuals.

In randomly selecting the constitution of the individuals we face, for the first time, one of the central issues of evolutionary computation: the viability of properly handling randomness in a digital computer. We shall dwell upon this matter at length in chapter 2. Let us simply say, at this point, that the randomness of the initial population is assumed without further consideration. If this is the case, we may validly ask: What are the chances that, by mere coincidence, one of the individuals represents the solution to the problem? In this particularly simple example such probability is easily calculated to be p=10/16=0.625 (If the solution were found in the numbers being encoded, between 0 and 15). In general, however, this probability is so close to 0 that any exhaustive ennumerative process is ruled out.

Table T.1.1.1.1. A population of 10 Individuals.

Number	Individual
1	0111
2	0100
3	0101
4	0001
5	0000
6	1100
7	1000
8	1001
9	1000
10	1010

1.1.1.3 Selection

Out of the 10 individuals which comprise our population we shall select a subset consisting of 8. These we select to be the first 8. Later on we shall see why this choice was made and other possible alternatives.

1.1.1.4 Evaluation

At this point we have 8 candidate solutions represented in the 8 chosen individuals. Are the candidate solutions "good"? It is here that we must determine whether an individual is fit. And it is here that the role played by the environment is explicitly imposed upon our problem. How well, we ask, does an individual (or, rather, the solution encoded in its bits) perform in terms of a given problem? Here we want to know whether the number encoded fares well when presented with $Y = 2X^2-3$ and the set of prescribed conditions.

In Table T.1.1.1.2. we show the results of this evaluation. We have added a column labelled "Decimal", where the numeric value of the encoded string is shown in base 10. We also added a column labelled "Fitness" where we show the value taken by the function for the said value. In this case "fitness" is easily equated to "Y" since we are simply trying to maximize the function.

Table T.1.1.1.2. A population of 8 Individuals and their Fitness.
$(Y = 2X^2-3)$

Number	Individual	Decimal	Fitness
1	0111	7	95.0000
2	0100	4	29.0000
3	0101	5	47.0000
4	0001	1	-1.0000
5	0000	0	-3.0000
6	1100	12	285.0000
7	1000	8	125.0000
8	1001	9	159.0000

1.1.1.5 Survival

Now is the time to decide on the survival of the fittest individuals. We choose the following strategy:

a) Normalize the fitness of each of the individuals. This is easily achieved by making

$$F_i = \frac{f_i}{\sum_{i=1}^{N} f_i} \quad \forall i \tag{1.1.1.a}$$

It is assumed that the population is of size *N*.

At this point one must make sure, for reasons that will become apparent shortly, that all fitnesses are positive. When one or more values are negative there are several ways to condition the data. We perform the following procedure:

1) Add up all the absolute values of the fitnesses.
2) Find the mean.
3) Obtain a constant value resulting from adding up the absolute value of the smallest (most negative) number and the mean.
4) Add this constant to each of the fitnesses.

This procedure is expressed in the following equation:

$$f_i' = f_i + \frac{1}{N}\sum_i |f_i| + |\min_i(f_i)| \quad \forall i \tag{1.1.1.b}$$

Equation (1.1.1.a) is, therefore, replaced by

$$F_i = \frac{f_i'}{\sum_{i=1}^{N} f_i'} \quad \forall i \tag{1.1.1.c}$$

The results of normalizing the data are shown in Table T.1.1.1.3.

Notice that in table T.1.1.1.3 we have also included a column labelled "CDF", where

we have registered the cumulative F_i. That is

$$CDF(i)=\sum_{j=1}^{i} F_j \qquad (1.1.1.d)$$

The usefulness of CDF(i) will become apparent in what follows.

We shall think of the individuals as being more apt to survive the larger their fitness. To establish a clear correspondence between fitness and survival we may think that we have a die with 10,000 faces and that, in this example, 2,533 of them are labelled "6"; 1,696 are labelled "8"; 1,469 are labelled "7", etc. In such case, if we roll our die a large number of times, the probability of getting a face labelled "6" will be 0.2533; the probability of getting a face labelled "8" will be 0.1696, etc.

Table T.1.1.1.3. A population of 8 Individuals and their CDF.

(Average Fitness=93)

Number	Individual	Decimal	Fitness	Normalized Fitness	CDF
6	1100	12	285.0000	381.0000	0.2533
8	1001	9	159.0000	255.0000	0.4229
7	1000	8	125.0000	221.0000	0.5698
1	0111	7	95.0000	191.0000	0.6968
3	0101	5	47.0000	143.0000	0.7919
2	0100	4	29.0000	125.0000	0.8750
4	0001	1	-1.0000	95.0000	0.9382
5	0000	0	-3.0000	93.0000	1.0000

Once we have labelled our hypothetical die the process of survival reduces to rolling the die and allowing the individuals whose numbers show up to survive. We shall roll the die N times (N=8) and obtain a new population P_1 (g=1) from the old population P_0 (g=0). The individuals from population P_{g-1} are replaced by the individuals of population P_g. The individuals of population g-1 are lost.

We aim at a way to simulate the behavior of a multi-faced die in the computer. By finding the CDF as above we may achieve this effect as follows:

1) Generate a random number "R" between 0 and 1 ($0 \leq R < 1$).
2) Set i $\leftarrow$ 1.
3) If CDF(i) is smaller than R, select the i-th individual; otherwise, set i $\leftarrow$ i+1 and repeat this step.

We assume that R is uniformly distributed (see section 2.3.1). If such is the case, it is easy to convince oneself that the method above will yield the desired result.

At this point it should be clear why it is necessary to normalize the fitness. Since, in fact, we are establishing a one-to-one correspondence between the fitness of the individual and the frequency with which such an individual is selected, it would make no sense to allow a negative value. Furthermore, a direct translation of axis (i.e. $f_i^{/} = f_i + |\min_i(f_i)|$) will not suffice since this would imply that the element with the least value would have a fitness of 0 and, consequently, a probability of being chosen equal to 0. It is, therefore, necessary to assign an arbitrary offset value to the least value. In our case we have chosen the offset to be the mean of the absolute values of the fitness ($\frac{1}{N}\sum_i |f_i|$).

We have simulated the rolling of the multi-faced die 8 times. The results are shown in table T.1.1.1.4.

Table T.1.1.1.4. A new population of 8 Individuals.

Number	Old Number	Individual
1	2	0100
2	8	1001
3	6	1100
4	6	1100
5	3	0101
6	8	1001
7	3	0101
8	6	1100

It is easy to see that, in this new table, certain individuals from the old population have been chosen several times, while others were not selected. Of course, for such a small N the observed frequency cannot approximate the expected probability.

What we have done is to allow the survival of the individuals in direct proportion to their fitness, as desired.

Calculating F_i from equation (1.1.1.a), we get table T.1.1.1.5. Notice that the average fitness for population 0 (the initial population) and population 1 has increased from 93 to 162.

1.1.1.6 Crossover

Once we have the surviving population we hypothesize that, somehow, the desirable characteristics of each individual are encoded in its genome[6]. Moreover, we believe that such encoded characteristics may be enhanced by combining the genes[7] of the individuals.

It is this exchange of genetic material which gives origin to the term "Genetic Algorithm". To achieve such a genetic recombination we propose the following algorithm:

1. Randomly select one individual from the population. Call this individual I_1.
2. Randomly select a second individual. Call it I_2.
3. Randomly select a number L ($1 \leq L < l-1$), where l = number of bits.
4. Exchange the rightmost L bits of I_1 and I_2; this gives rise to 2 possibly different individuals N_1 and N_2.
5. Replace the old individuals I_1 and I_2 with N_1 and N_2.

Table T.1.1.1.5. A new population with fitness.
(Average Fitness=162)

Number	Old Number	Individual	Decimal	Fitness
8	6	1100	12	285.0000
4	6	1100	12	285.0000
3	6	1100	12	285.0000
6	8	1001	9	159.0000
2	8	1001	9	159.0000
7	3	0101	5	47.0000
5	3	0101	5	47.0000
1	2	0100	4	29.0000

[6] By "genome" we mean the bits which conform the individual.

[7] By "gene" we mean each of the individual bits comprising an individual.

This process is repeated such that N new individuals replace the old population's individuals.

The results of applying this procedure are shown in table T.1.1.1.6. In this table we show the individuals which give rise to the new individuals as well as the crossover locus and the new fitness. In table T.1.1.1.7. we show the same population ordered according to the new fitnesses.

Notice that the best individual of the new population has, indeed, a better fitness than any of its predecessors. As an example, consider individuals 1 and 2 in table T.1.1.1.6. The indicated crossover locus is 2. Hence, the genetic material is exchanged as shown in figure F.1.1.1.2.

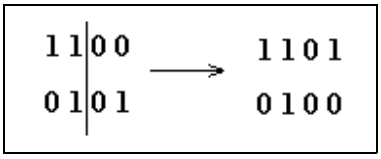

Figure F.1.1.1.2. Crossover.

Table T.1.1.1.6. Population after Crossover.

Number	Old	Old	Locus	New Indi	Decimal	Fitness
1	6	1100	2	1101	13	335.0000
2	3	0101	2	0100	4	29.0000
3	8	1001	2	1000	8	125.0000
4	2	0100	2	0101	5	47.0000
5	8	1001	1	1001	9	159.0000
6	8	1001	1	1001	9	159.0000
7	3	0101	2	0101	5	47.0000
8	8	1001	2	1001	9	159.0000

1.1.1.7 Mutation

The individuals of the new population result from the genetic recombination of the prior population's individuals. In principle, we expect the new individuals to achieve better fitness than the old ones, as in the example. However, even if such were the case it is clear that, in terms of string components, we have added nothing new. Most important is the possibility of introducing a slight chance of error in the reproduction process.

Nature bases evolution in the very small chance that a coding error in an individual genome will result in a better individual once in a very long while. We shall mimic such a fortuitous event by randomly altering the value of set of randomly selected bits in the population. The probability P_m with which this happens is, usually, very low.

Table T.1.1.1.7. Population after crossover ordered by fitness.

Number	Old	Old	Locus	New	Decimal	Fitness
1	6	1100	2	1101	13	335.0000
8	8	1001	2	1001	9	159.0000
6	8	1001	1	1001	9	159.0000
5	8	1001	1	1001	9	159.0000
3	8	1001	2	1000	8	125.0000
7	3	0101	2	0101	5	47.0000
4	2	0100	2	0101	5	47.0000
2	3	0101	2	0100	4	29.0000

In table T.1.1.1.8. we show the results of applying the mutation procedure to population P_1. For the purposes of this example we have chosen P_m=0.1, which is much larger than usual. Accordingly two bits were mutated. These have been boldfaced in the table.

With a certain amount of luck[8], both mutations[9] hit upon ’0’ and transformed individuals with decimal values "5" and "8" into values "13" and "10" respectively. The associated fitnesses changed from 47 and 125 to 335 and 197, respectively .

[8] Actually, the probability of this happening is

$$\begin{aligned} P_{00} &= P_0 \times P_0 \\ &= \frac{17}{32} \times \frac{16}{32} \\ &\approx 0.2656 \end{aligned}$$

[9] The number of mutations is, itself, a random variable. Since there are 32 bits in this population, the expected number of mutations per generation is 3.2.

Table T.1.1.1.8. Population P_1 after mutation.
($P_m = 0.1$)

Number	Old Individual	New Individual	Fitness
7	**0**101	**1**101	335.0000
1	1101	1101	335.0000
3	10**0**0	10**1**0	197.0000
5	1001	1001	159.0000
6	1001	1001	159.0000
8	1001	1001	159.0000
4	0101	0101	47.0000
2	0100	0100	29.0000

It is important to stress the fact that, usually, mutations are not as effective as the previous example would lead us to believe. In this case an important part of the displayed effectiveness is due to the nature of the function we have been using.

1.1.2 A Genetic Algorithm

There are some issues which we have no yet considered. For instance, we have assumed that all individuals are to undergo a process of crossover. A more general case would be to allow crossover with some probability $P_c > 0$. When $P_c = 1$ we have the case already discussed. Another issue is the one pertaining the number of descendants that each individual may have. These and other interesting issues will be considered later in the text.

Nevertheless, even in this restricted scenario, a genetic algorithm may now be defined, as follows:

ALGORITHM A.1.1.2

1. Set i=0 and generate an initial population P_i (P_0).
2. Select a subset of P_i.
3. Evaluate the individuals of P_i according to some fitness criteria.

4. Test for some convergence criteria. If any of these is met, stop.
5. Allow the survival of a selected subset of individuals according to their fitnesses.
6. Exchange the genetic material between the surviving members of P_i.
7. Mutate some of the genes of P_i.
8. Set $i \leftarrow i+1$ and proceed with step 2.

Notice that this algorithm has been kept independent of which subset will be selected in step 1, which fitness criteria will be set in step 2, which convergence tests will be performed in step 3, how we pick the surviving individuals in step 4, how the crossover is to be performed in step 5 and, finally, how we mutate the genes during step 6. In so doing we have defined a general genetic algorithm. If, however, all of the above steps are performed as described in the example above, then we have what we, throughout this text, shall call a Simple Genetic Algorithm (SGA).

If we allow the SGA to proceed and if we establish as our only stopping criterion a prescribed number of iterations ι, where $\iota=5$ the algorithm will yield the results shown in table T.1.1.1.9.

In the table the column heading "#" corresponds to the order in which the individual appears during the generation; the column heading "Ind." stands for the encoding of the individual; the column heading "Fit" shows the unnormalized fitness.

The best 6 individuals for the five generations are shown in figure F.1.1.1.3. Notice that during the first generation the best individual is far from the best value. By the third generation one individual has already reached the optimum value. Best individuals are found in the fourth and fifth generations. By the fifth generation four of the eight individuals (four of the six shown) have reached the best value.

Thus, with this simple example we have illustrated the essentials of a genetic algorithm. Several questions now arise immediately. For instance: Is this process always better than a random walk? How do we pick the size of the population? What probability of mutation is best? Does the algorithm always converge to the best values?

We shall try to analyze these and other related questions as we proceed in the text. In some cases we will be able to provide reasonable answers. In some others we will not be in such position. The genetic algorithmic process is a complex phenomenon and there are still many unanswered questions. However, the usefulness of the method is attested by the wealth of works under way. An interesting number of works in this area have been and are being published in several journals and conferences throughout the world. GAs are still in the process of being properly understood. The method requires of the interplay of many disciplines: computer science, statistics, biology, numerical analysis, etc. In the following section we review some of the subjects we shall be dealing with in order to try to understand and apply the genetic algorithmic tools.

Table T.1.1.1.9. Genetic process in 5 Generations.

Generation 1			Generation 2			Generation 3			Generation 4			Generation 5		
#	Ind.	Fit	#	Ind.	Fit	#	Ind.	Fit	#	Ind.	Fit	#	Ind.	Fit
6	1100	285	2	1101	335	1	1111	447	3	1111	447	4	1111	447
8	1001	159	1	1101	335	4	1101	335	2	1111	447	3	1111	447
7	1000	125	6	1001	159	3	1101	335	1	1111	447	2	1111	447
1	0111	95	5	1001	159	2	1101	335	5	1101	335	1	1111	447
3	0101	47	4	1001	159	8	1001	159	4	1101	335	5	1011	239
2	0100	29	3	1001	159	7	1001	159	6	1011	239	6	1001	159
4	0001	-1	7	1000	125	6	1001	159	7	1001	159	8	0011	15
5	0000	-3	8	0100	29	5	1001	159	8	0101	47	7	0011	15

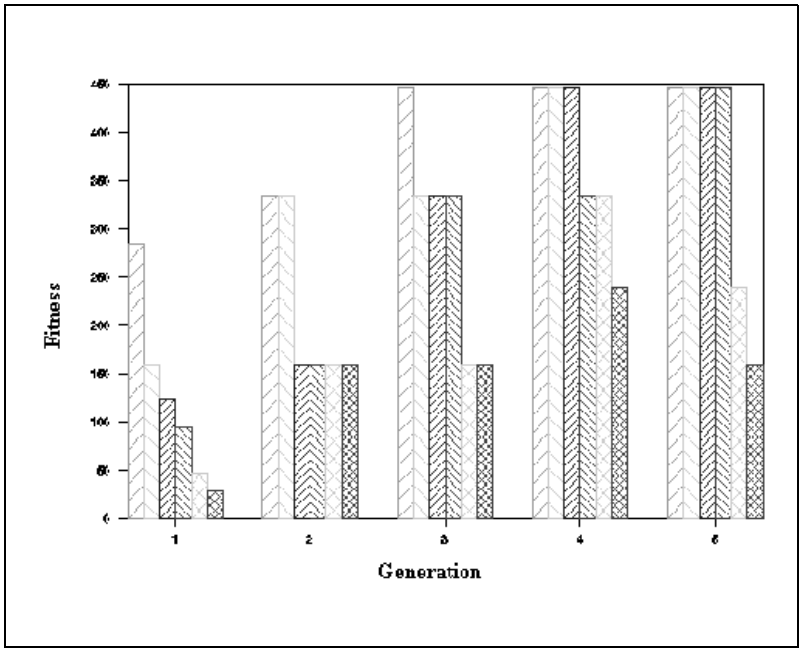

Figure F.1.1.1.3. Genetic Process.

4 Theoretical Considerations

Seeking the Truth.

The student came to the Zen master and said:
"I am seeking the truth. In what state of mind should I train myself, so as to find it?"
There is no mind, so you cannot put it in any state. There is no truth, so you cannot train yourself for it.
"Oh, how can you lie like this?"
I have no tong to talk to others. How can I lie to you?
"I cannot follow you. I cannot understand you."
I cannot understand myself, said the master.

Koan, *Zen Buddhism*

The one who knows the TAO
Does not talk about it.
The one who talks about the TAO
Does not know it.

Tao Teh King

Genetic Algorithms are a tool of our technological era. Their existence is a direct consequence of the admixture of genetic knowledge, on the one hand, and computer development, on the other. It should come as no surprise that the exact nature of their behavior is, for the best part, still unknown. In other words, although there exists ample and mounting evidence as to their effectiveness as a very practical tool to attack otherwise intractable problems, there is, as of today, not an accepted closed theory that explains them fully. The pioneering efforts of Holland were, for the most part, directed to obtaining such theory. He achieved such a goal only partially, in terms of what he called *schemas*. These schemas allow a quantitative measure of a GA's behavior only as a function of very broad considerations and the results obtained are of relative practical significance. Furthermore, the mentioned results are applicable only to the Simple Genetic Algorithm. Much of the effort of the present day researchers has been, and is being, focused at attempting a generalization of those initial results and on alternative tools to achieve a broader and better understanding of the way GAs will behave when faced with different problems.

On dwelling with the problem of why GAs work one faces the problem of precisely defining which of the many variations of a GA we are really attempting to analyze. In fact, a main problem of GA students is that, in many cases, one fails to recognize the

fact that there *are* many such variations. In the following sections we analyze a methodology to establish a systematic classification of the variations just mentioned. We also review some of the classic results of schema theory and point out some interesting recent alternatives.

4.2 The Concept of Schema

Schemas and their properties are interesting notational devices for rigorously discussing and classifying string similarities. A schema is a template describing a subset of strings with similarities at certain positions. It is defined over the alphabet {0,1,*}. For example, the schema *0000 matches two strings: {00000 and 10000}. The schema *111* matches the strings {01110, 01111, 11110, 11111}.

There are $(k+1)^l$ possible schemas for an alphabet which consists of k symbols and of length l. For k=2 and l=5 (as in the examples above) there are $3^l = 3^5 = 243$ schemas.

How much information do we get by examining the schemas? This is related to the number of unique schemas contained in the population. A particular string represents 2^l schemas. Therefore, a population of size n contains N (between 2^l and $n \cdot 2^l$) schemas.

$$2^l \leq N \leq n \cdot 2^l$$

Of the N schemas in a population, how many are processed? It turns out [GOLD89a] that the number is approximately n^3. There are, at the same time, n function evaluations. The ratio of function evaluations to processed schemas η is n^2 and this characteristic is called *implicit parallelism*.

4.2.1 Schema Order

The order of a schema H denoted by o(H) is simply the number of fixed positions (in a binary alphabet, the number of 0's and 1's) present in the schema. For instance,

H = 011*1** o(H) = 4
H = 0****** o(H) = 1
H = 1*1*1** o(H) = 3

4.2.2 Defining Length

The defining length of schema H denoted by $\delta(H)$ is the distance between the first and last specific bit position. For instance,

H = 011*1** $\delta(H)$ = 5-1 = 4
H = 0****** $\delta(H)$ = 1-1 = 0
H = 1*1*1** $\delta(H)$ = 5-1 = 4

4.2.3 Selection

At time t there are *m* instances of a particular schema H within the population A(t) where we write

$$m = m(H,t) \tag{4.2.3.a}$$

During selection, a string is copied according to its fitness, i.e. string A_t is selected with probability

$$p_t = \frac{f_t}{\sum f_t} \tag{4.2.3.b}$$

(This follows from the *proportional selection* strategy defined for the SGA. If the selection process is not defined as such, this consideration does no longer apply).

After picking a nonoverlapping population of size n with replacement from A(t), we have:

$$m(H,t+1) = m(H,t)\cdot n \cdot \frac{f(H)}{\sum f_t} \tag{4.2.3.c}$$

where f(H) is the average fitness of the strings representing schema H at time t.

The average fitness of the entire population may be written as $\bar{f} = \frac{\sum f_t}{n}$ and we may rewrite the reproductive schema growth equation as follows:

$$m(H,t+1) = m(H,t)\frac{f(H)}{\bar{f}} \tag{4.2.3.d}$$

That is, a particular schema grows as the ratio of the average fitness of the schema to the average fitness of the population. Therefore, the number of above-average schema grows, while the number of below-average schema decreases.

Let us assume that a particular schema H remains above average an amount $c\bar{f}$ with c constant. Then the schema difference equation may be written as:

$$m(H,t+1) = m(H,t)\frac{(\bar{f}+c\bar{f})}{\bar{f}} = m(H,t)(1+c)$$

Starting at t=0 and assuming a stationary value of c, we get:

$$m(H,t) = m(H,0)\ (1 + c)^t \tag{4.2.3.e}$$

The effect of selection is now quantitatively clear: selection allocates exponentially

increasing (decreasing) numbers of trials to above-(below)average schemas.

4.2.4 Crossover

Crossover is a structured yet randomized information exchange between strings. It creates new structures with a minimum of disruption to the allocation strategy dictated by selection alone.

Let us assume the following string and the corresponding schemas:

$$\begin{aligned} A &= 0\,1\,1 \mid 1\,0\,0\,0 \\ H_1 &= *\,1\,* \mid *\,*\,*\,0 \\ H_2 &= *\,*\,* \mid 1\,0\,*\,* \end{aligned}$$

where the "|" signs the hypothetical crossover point. It is clear that schema H_1 is less likely to survive, in general, than schema H_2.

Note that $\delta(H_1)=5$, while $\delta(H_2)=1$. There are l-1 possible crossover sites (i.e. 7-1 = 6). Therefore, H_1 is destroyed with probability

$$p_d = \frac{\delta(H_1)}{l-1} = 5/6$$

More generally, survival probability under this kind of crossover is[10]

$$p_s \geq 1-\frac{\delta(H)}{(l-1)} \qquad (4.2.4.a)$$

If crossover is itself performed by random choice with probability p_c, then we have

$$p_s \geq 1-p_c\frac{\delta(H)}{l-1} \qquad (4.2.4.b)$$

which reduces to the previous expression when $p_c = 1$.

The combined effect of selection and crossover may now be considered, as follows:

---

[10]The "≥" is due to the fact that, when considering schema disruption we must take into account that the schema under consideration will sometimes be created from different schemas. That is, when crossover disrupts other schemas it may create copies of "our" schema.

$$m(H,t+1) \geq m(H,t)\cdot\frac{f(H)}{\bar{f}}\left[1-p_c\frac{\delta(H)}{l-1}\right] \qquad (4.2.4.c)$$

As outlined in section 1.2.2. there are several ways to perform crossover. The schema analysis performed above is particular to 1-point crossover. Next we examine, briefly, crossover in the *2-point* and *uniform* variations.

4.2.4.1 Two Point Crossover

There are exactly $\binom{l}{2}$ ways to select two points of cut in a string of length $l > 1$ if we consider it as joined in its end points. Assume that in this string we may find a schema of defining length $\delta(H)$ and order $o(H)$. If $o(H) = 2$ then H breaks down when one of the points of cut lies within the segment determined by $\delta(H)$ and the other outside of it. The probability of disruption of a schema of order 2 is, therefore:

$$\begin{aligned} P_{\delta(H),2} &= \frac{\delta(H)(l-\delta(H))}{\binom{l}{2}} \\ &= \frac{\delta(H)(l-\delta(H))}{l(l-1)} \end{aligned} \qquad (4.2.4.1.a)$$

If, on the other hand, we assume that the schema has all the places within $\delta(H)$ defined (that is, $o(H) = \delta(H) + 1$, then the schema is also disrupted when the two points of cut fall within the segment considered by $\delta(H)$. We may modify (4.2.4.1.a) and get

$$P_{\delta(H),\delta(H)-1} = \frac{\delta(H)(l-\delta(H)) + \binom{\delta(H)}{2}}{\binom{l}{2}} \qquad (4.2.4.1.b)$$

In general,

$$P_{\delta(H)} \leq P_d \leq P_{\delta(H),\delta(H)-1} \qquad (4.2.4.1.c)$$

4.2.4.2 Uniform Crossover

Let A and B be two strings which are to be crossed over to form a new couple C and D. In this kind of crossover each position of chain C is randomly selected from A and B. The winning (selected) bits will be "taken" so that they will occupy the same places of C as they originally did in A or B; the losing bits will be "taken" to D.Assume that we wish to preserve a schema H of which one of the parent chains is an instance. This schema has $o(H)$ fixed bits regardless to which of the offspring the first fixed bit of H is taken. What *does* matter is that the remaining $o(H) - 1$ fixed bits in the schema are taken to the same offspring. Each one of these bits has a probability $\frac{1}{2}$ of staying in the same string as the first one (assuming statistical independence). Hence, the probability that all bits from the schema are copied to the same descendant is $\left(\frac{1}{2}\right)^{o(H)-1}$. Therefore, the probability of disrupting schema H is

$$P_{\delta(H),o(H)} = 1 - \left(\frac{1}{2}\right)^{o(H)-1} \qquad (4.2.4.2.a)$$

The above is the worst case, i.e. when the parent strings do not agree in any of the positions of interest for the schema. In general:

$$P_d \leq 1 - \left(\frac{1}{2}\right)^{o(H)-1} \qquad (4.2.4.2.b)$$

4.2.5 Mutation

Mutation is the random alteration of a single position with probability p_m. For a schema to survive all of the alleles[11] must remain unaltered. This probability of survival under mutation is, of course 1-p_m for a single allele and the probability of survival of a given schema is

$$p_{s_m} = (1-p_m)^{o(H)} \qquad (4.2.5.a)$$

If $p_m \ll 1$ we may write, after expanding (4.2.5.a) and disregarding terms of higher degree;

$$p_{s_m} \approx 1-o(H)p_m \qquad (4.2.5.b)$$

4.3 The Fundamental Theorem of Genetic Algorithms

[11]The different possible settings for a trait (e.g. blue, brown, hazed) are called *alleles*. Very roughly, one can think of a gene as encoding a *trait*.

From the above we conclude that a schema "H" receives an expected number of copies as follows (ignoring elements of low order):

$$m(H,t+1) \geq m(H,t)\cdot\frac{f(H)}{\bar{f}}\left[1-p_c\frac{\delta(H)}{l-1}-o(H)p_m\right] \qquad (4.3.a)$$

This result is known as the *Fundamental Theorem of Genetic Algorithms*. This theorem refers, specifically, to those GAs arising from an SGA strategy for reasons that should be fairly obvious to the reader.

The *fundamental theorem* is generally interpreted by observing that short, low-order schema (usually called *building blocks*) give rise to longer, higher order schema[12]. Furthermore, as seen from (4.2.3.e), this happens while giving exponentially increasing trials to these building blocks. This observation results from analyzing the way the SGA works in general.

4.4 Genetic Algorithms as a Self-Adaptive Optimizing Method

The discussion above allows us to determine how schemas are passed on from a generation to the next one. We also concluded that above-average schemas receive exponential number of trials during the algorithm's development. The question now is: Is this a good thing? In what follows we try to answer this question by analyzing the "2-armed bandit"[13] problem and its generalization, the "k-armed bandit" problem. These problems typify a set of problems which necessitate of self-adaptive strategies in order to be solved successfully. We shall show that these strategies give rise to the need of exponential sampling of hypothetical solutions. Furthermore, we shall also argue that a genetic algorithm constitutes an embodiment of such strategies.

4.4.1 The Two-armed Bandit

Suppose we have a two-armed slot machine[14]. In this machine we have a fixed probability of winning a certain reward when activating arm 1 and a certain different mean reward when activating arm 2. We denote arm's 1 expected award μ_1 and we call its variance σ_1^2. Likewise we denote arm's 2 parameters by μ_2 and σ_2^2. Further, we know that $\mu_1 \neq \mu_2$. Our aim is to maximize our gains or, alternatively, to minimize our losses.

However, we do not know a priori which of μ_1 and μ_2 is the larger of the two. In fact,

---

[12]This is called the *Building Block Hypothesis* or *BBH*.

[13]Colloquially a typical one armed slot machine is called a *one-armed bandit*.

[14]We will denote the Two-Armed Bandit by *TAB*.

we do not know any of μ_1, μ_2, σ_1, σ_2. We propose the following strategy. Determine two phases: an exploration phase and an exploitation phase. During the exploration phase we shall try to determine which is the best arm. During the exploitation phase we shall try to take advantage of the knowledge obtained in phase 1 and gain as much as we can assuming that we were able to determine correctly which of arms 1 or 2 is the best.

We, therefore, propose the following:

- Establish a priori a global fixed number of trials to both arms (say N).
- Exploration: Allocate n trials to each arm (where 2n < N).
- Exploitation: Allocate the remaining N-2n trials to the determined best arm.

Let us call the arm with the observed higher payoff A_h; let us call the arm with the observed lower payoff A_l. Assuming we know N, μ_1 and μ_2, where μ_1 is the best arm's mean award, we may calculate the expected loss as

$$L(N,n) = (\mu_1 - \mu_2)\cdot[(N-n)q + n(1-q)] \qquad (4.4.1.a)$$

where q is the probability that the observed best arm is actually the worst arm. Our goal is to find $n = n^*$ that minimizes L(N - *n*, *n*). This can be done by taking the derivative of L(N - *n*, *n*) with respect to *n*, setting it to zero and solving for *n*. We get,

$$\frac{dL}{dn} = (\mu_1 - \mu_2)\left(1 - 2q + (N - 2n)\frac{dq}{dn}\right) = 0 \qquad (4.4.1.b)$$

We need to express *q* in terms of *n* so that we can find *dq/dn*. *q* is the probability that observed worst arm *l* is the best arm (arm 1), or $q = Pr(A_l(N - n) = A_1)$. We are assuming that indeed $A_l(N - n)$ is A_1. Then A_1 was given *n* trials. Let S_1^n be the sum of the payoff of the *n* trials given to A_1. Also, let S_2^{N-n} be the sum of the payoffs of the N - *n* trials given to A_2. Then

$$q = Pr\left(\frac{S_2^{N-n}}{N-n} > \frac{S_1^n}{n}\right) \qquad (4.4.1.c)$$

that is, the probability that the observed average payoff of A_2 is higher than that of A_1. Equivalently,

$$q = Pr\left(\left(\frac{S_1^n}{n} - \frac{S_2^{N-n}}{N-n}\right) < 0\right) \qquad (4.4.1.d)$$

This probability may be estimated from the tail of the normal distribution[15].

[15]See section 2.3.2.

$$q(n) \approx \frac{1}{\sqrt{2\pi}} \frac{e^{-x^2/2}}{x} \text{ where } x = \frac{\mu_1 - \mu_2}{\sqrt{\sigma_1^2 + \sigma_2^2}} \sqrt{n} \qquad (4.4.1.e)$$

And, according to Frantz [MITC96, pp. 120-122], the optimal allocation of trials n^* to the observed second best of the two random variables corresponding to the two-armed bandit problem is approximated by

$$n^* \approx c_1 \ln\left(\frac{c_2 N^2}{\ln(c_3 N^2)} \right) \qquad (4.4.1.f)$$

where c_1, c_2 and c_3 are positive constants. The details of the solution are not as relevant as the form. If we rearrange the terms of (4.4.1.f) to get an expression for $N - n^*$, the optimal allocation of trials to the observed better arm is

$$N - n^* \approx e^{n^*/2c_1} \sqrt{\frac{\ln(c_3 N^2)}{c_2}} - n^* \qquad (4.4.1.g)$$

As n^* grows the exponential dominates and, letting $c = 1/2c_1$, we may further approximate by

$$N - n^* \approx e^{cn^*} \qquad (4.4.1.h)$$

In other words, to allocate trials optimally we should give slightly more than exponentially increasing trials to the observed best arm.

The strategy just mentioned is normally unrealizable since it assumes knowledge of the outcomes before they occur. However, a method which approaches the ideal allocation problem is the SGA. The schema theorem guarantees giving at least an exponentially increasing number of trials to the observed best building blocks. In this way, the SGA is a realizable and nearly optimal procedure.

In the SGA we are no longer solving a simple two-armed problem. Rather, we consider the simultaneous solution to many multi-armed bandits. We shall demonstrate that the SGA may be thought of as the composition of many k-armed bandit problems.
...

4.4.3 The Minimal Deceptive Problem

The question we now pose is the following: Which problems are difficult for a SGA to process? In terms of the *BBH* we look for those cases where short, low-order building blocks lead to incorrect longer, higher order building blocks. The smallest problem we can analyze is a 2-bit problem.

Suppose we have the following set of 4 order-2 schemas over two defining positions as follows:

$$
\begin{array}{ll}
* * * 0 * * * * * 0 * & f_{00} \\
* * * 0 * * * * * 1 * & f_{01} \\
* * * 1 * * * * * 0 * & f_{10} \\
* * * 1 * * * * * 1 * & f_{11} \\
\text{-----}\delta(H)\text{----} &
\end{array}
$$

The fitness values (f_{xx}) are schema averages, assumed to be constant with no variance.

Let us start by assuming that

$$f_{11} > f_{00};\ f_{11} > f_{01};\ f_{11} > f_{10} \tag{4.4.3.a}$$

We want a problem where one or both of the suboptimal order-1 schemas are better than the optimal order-1 schemas, i.e. we want one or both of the following conditions to hold:

$$f(0*) > f(1*) \tag{4.4.3.b}$$
$$f(*0) > f(*1) \tag{4.4.3.c}$$

In these expressions we have dropped consideration of all alleles other than the two defining positions and the fitness expression implies an average over all strings contained within the specified similarity subset. Thus we want:

$$\frac{f(00)+f(01)}{2} > \frac{f(10)+f(11)}{2} \tag{4.4.3.d}$$
$$\frac{f(00)+f(10)}{2} > \frac{f(01)+f(11)}{2} \tag{4.4.3.e}$$

Both expressions may not hold simultaneously and, without loss of generality, we take the first expression to hold.

From these we recognize that there are two types of minimal deceptive problems:

Type I: $f_{01} > f_{00}$
Type II: $f_{00} > f_{01}$

Figures F.4.4.3.1. and F.4.4.3.2. represent sketches of these problems.

4.4.3.1 Extended Schema Analysis for the Two-Problem

From the schema theorem we expect to find difficulty when

$$\frac{f(11)}{\bar{f}}\left[1 - p_c \frac{\delta(11)}{l-1}\right] \leq 1 \qquad (4.4.3.1.a)$$

(assuming $p_m = 0$). A more careful analysis leads us to consider crossover more closely.

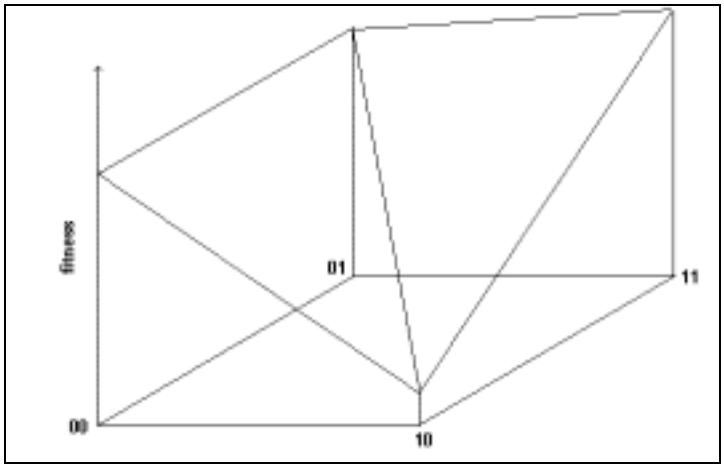

Figure F.4.4.3.1. Sketch of Type I minimal deceptive problem ($f_{01} > f_{00}$)

We lose the schema whenever a cross occurs between the schema's outermost defining bits. A full crossover yield table (T.4.4.3.1.1.) is shown, where an S is used to indicate that the offspring are the same as their parents.

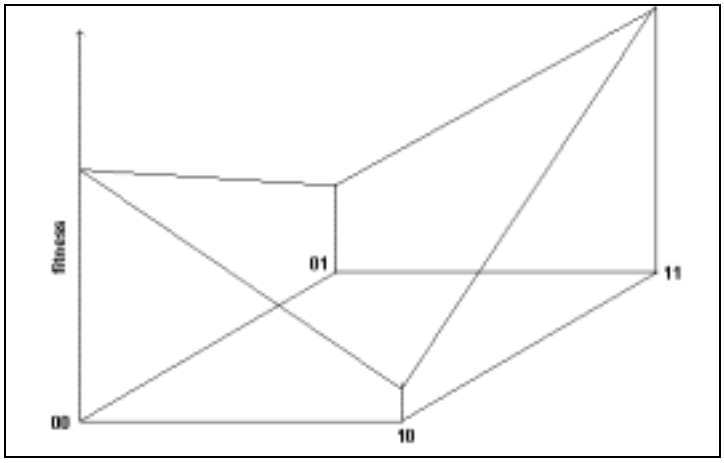

Figure F.4.4.3.2. Sketch of Type II minimal deceptive problem ($f_{00} > f_{01}$)

Table T.4.4.3.1.1.

	00	01	10	11
00	S	S	S	01, 10
01	S	S	00, 11	S
10	S	11, 00	S	S
11	10, 01	S	S	S

Assuming proportional selection, 1-point crossover and random mating we obtain the following autonomous nonlinear difference equations:

$$
\begin{aligned}
P_{11}^{t-1} &= P_{11}^{t}\cdot\frac{f_{11}}{\bar{f}}\left[1-p_c\frac{f_{00}}{\bar{f}}P_{00}^{t}\right] + p_c\frac{f_{01}f_{10}}{\bar{f}^2}P_{01}^{t}P_{10}^{t}\\
P_{10}^{t-1} &= P_{10}^{t}\cdot\frac{f_{10}}{\bar{f}}\left[1-p_c\frac{f_{01}}{\bar{f}}P_{01}^{t}\right] + p_c\frac{f_{00}f_{11}}{\bar{f}^2}P_{00}^{t}P_{11}^{t}\\
P_{01}^{t-1} &= P_{01}^{t}\cdot\frac{f_{01}}{\bar{f}}\left[1-p_c\frac{f_{10}}{\bar{f}}P_{10}^{t}\right] + p_c\frac{f_{00}f_{11}}{\bar{f}^2}P_{00}^{t}P_{11}^{t}\\
P_{00}^{t-1} &= P_{00}^{t}\cdot\frac{f_{00}}{\bar{f}}\left[1-p_c\frac{f_{11}}{\bar{f}}P_{11}^{t}\right] + p_c\frac{f_{01}f_{10}}{\bar{f}^2}P_{01}^{t}P_{10}^{t}
\end{aligned}
\qquad (4.4.3.1.b)
$$

The variable $\bar{f}$ is simply generation t's population average fitness, which may be evaluated from

$$\bar{f} = P_{00}^{t}f_{00} + P_{01}^{t}f_{01} + P_{10}^{t}f_{10} + P_{11}^{t}f_{11} \qquad (4.4.3.1.c)$$

The parameter $p_c^{\,t}$ is the probability of having a cross that falls between the two defining bits of the schema:

$$p_c^{\,t} = p_c\cdot\frac{\delta(H)}{l-1} \qquad (4.4.3.1.d)$$

These equations predict the expected proportions of the four schemas in the next generation. With initial conditions we may follow the behavior of the schemas. A necessary condition for the convergence of the GA is that the expected proportion of

optimal schemas must go to unity in the limit, i.e. $\lim_{t\to\infty} P_{11}^{t} = 1$.

...

4.6 Hill Climbers

Hill Climbers (HCs) are a special kind of evolutionary computation variations where a given genome (a string or set of strings) is systematically modified and tested versus an objective function with the special purpose of finding an optimum value. For example, assume that we are given the genome G_i = *010110* and we set i=1. Assume, also, that we wish to minimize the function $Y = 2X^2 - 3$. Further, assume that the genome is to be interpreted as a weighted binary number (N = 22). In order to minimize this function we will systematically modify the bits of G_i, evaluate the objective function, set i ← i+1 and obtain a new G_i. We do this a specified number of times and keep the individual which yields the best (minimum value) fitness. The general Hill Climber algorithm is as follows:

ALGORITHM A.4.6.1

```
t:= 0
initialize H(t)
evaluate H(t)
while not terminate do
   H'(t):= variation [H(t)]
   evaluate [H'(t)]
   H(t+1) = best[ H(t), H'(t)]
   t:=t+1
enddo
```

Notice that this algorithm is similar to algorithm A.3.1. except that only one individual is considered. The *variation* part of the algorithm is simply a change in one of the bits of the genome.

In a way, a Hill Climber is the utmost of mutation. Essentially what we are doing is to mutate as best we can the genome of a given individual. The biological counterpart of a Hill Climber is an asexual unicellular individual, such as a bacteria.

The term "Hill Climber" comes from the belief that, once we are in the vicinity of a local optimum (*hill*), this kind of algorithm will efficiently zeroin (*climb*) on the best local optimum. Somewhat surprisingly, hill climbers have shown to be effective tools that sometimes might outperform genetic algorithms. In section 4.11. we incorporate an HC into the body of a GA.

In the following sections we give three variations of HCs.

4.6.1 Steepest Ascent Hill Climbing

1. Choose a string at random. Call this the *current hilltop*.
2. Going from left to right, systematically flip each bit in the string, recording the fitnesses of the resulting strings.
3. If any of the strings gives a fitness increase, set *current hilltop* to the best string.

Ties are decided at random.

4. If there is no fitness increase save *current hilltop* and go to step 1; else go to step 2.
5. When a set number of function evaluations is reached return the *hilltop*.

The following is one possible Xbase coding of the Steepest Ascent hill climber.
It is assumed that the string is selected from a previously existing table.

4.6.2 Next-Ascent Hill Climbing

1. Choose a string at random. Call this the current hilltop.
2. For i from 1 to *l* (the length of the string) flip bit i; if this results in a fitness increase keep the new string; otherwise flip i back. When an increase occurs, set this to current hilltop. Go to step 2 with the current hilltop but continue mutating the new string starting at the next sequential position in the string.
3. If no increases are found go to step 1.
4. Stop when a predetermined number of function evaluations is reached.

The following is one possible Xbase coding of the Next Ascent hill climber.
It is assumed that the string is selected from a previously existing table.

4.6.3 Random-mutation Hill Climbing

1. Choose a string at random. Call this the current hilltop.
2. Choose a locus at random. Flip the bit. If this leads to a better hilltop, call this the current hilltop.
3. Go to step 2 until an optimum string has been found or until a maximum number of function evaluations has been performed.
4. Return the best hilltop.

The following is one possible Xbase coding of the Random Mutation hill climber.
It is assumed that the string is selected from a previously existing table.

4.7 Royal Roads

The Building Block Hypothesis states that crossover combines short, observed high-

performance schemas into increasingly fit candidate solutions, but does not give any detail as to how this occurs.

The BBH suggests two features of fitness landscapes which are relevant to genetic algorithms:

a) The presence of short, low-order, highly fit schemas.
b) The presence of intermediate "stepping stones" (Intermediate-order higher-fitness schemas that result from the lower-order schemas and that, in turn, can combine to create even higher fitness-schemas).

Mitchell, Forrest and Holland designed a class of functions called *Royal Roads* that were meant to capture the essence of building blocks in idealized form.

A fitness function that explicitly contains these features (Royal Road R_1) is shown next. R_1 is defined using a list of schemas s_i. Each s_i is given with a coefficient c_i. The fitness $R_1(x)$ of a bit string x is defined as:

$$R_1(x) = \sum_i c_i \delta_i(x), \text{ where } \delta_i(x) = \begin{cases} 1 & \text{if } x \in s_i \\ 0 & \text{otherwise} \end{cases} \qquad (4.7.a)$$

For example, if x is an instance of exactly two of the order-8 schemas, $R_1(x) = 16$. Likewise $R_1(111...1) = 64$.

```
s1   = 11111111************************************************************; c1=8
s2   = ********11111111****************************************************; c2=8
s3   = ****************11111111********************************************; c3=8
s4   = ************************11111111************************************; c4=8
s5   = ********************************11111111****************************; c5=8
s6   = ****************************************11111111********************; c6=8
s7   = ************************************************11111111************; c7=8
s8   = ********************************************************11111111****; c8=8
sopt = 1111111111111111111111111111111111111111111111111111111111111111;
```

Figure F.4.7.1. An optimal string broken up into eight building blocks.

The function $R_1(x)$ (where x is a bit string) is computed by summing the coefficients c_s corresponding to the given schemas of which x is an instance.

For example:

$R_1(111111110...0) = 8$
$R_1(111111110...011111111) = 16$

Given the BBH, one might expect that the building block structure of R_1 will lay out a *royal road* for the GA to follow to the optimal string.

One might also expect that the GA will outperform Hill-Climbing schemes since a large number of bit positions must be optimized simultaneously in order to move from an instance of a lower-order schema (e.g. 11111111**...*) to an instance of a higher order intermediate schema (e.g. 11111111********11111111**...*).

4.7.1 Experimental Results[16]

A GA was run on R_1 with population size 128. *Sigma Truncation* selection was used to assign the expected number of offspring to each individual. In this scheme, each individual i's expected number of offspring is

$$1+(f_i-\bar{f})/2\sigma \qquad (4.7.1.a)$$

where F_i is *i*'s fitness; $\bar{f}$ is the mean fitness of the population and σ is the standard deviation of the fitness of the population. The number of expected offspring of any string was cut off at 1.5 (if the above formula gave a higher value, the value was reset to 1.5). This is a strict cutoff since it implies that most individuals will reproduce only 0, 1 or 2 times.

The effect of this selection scheme is to slow down convergence by restricting the effect that a single individual can have on the population regardless of how much fitter it is than the rest of the population.

The single point crossover rate was 0.7 per pair of parents and the bitwise mutation rate was 0.005.

The GA's performance on R_1 was compared to those of three different iterated hill-climbers:

a) Steepest Ascent Hill Climber
b) Next Ascent Hill Climber
c) Random Mutation Hill Climber

200 runs of each algorithm were performed. Each run used a different random seed. In each run the algorithm was allowed to run until an optimum was found. The total number of function evaluations was recorded. The mean and median for the number of function evaluations are given in the next table.

Table T.4.7.1.1. Comparison between a Genetic Algorithm and Hill Climber Variations for a Royal Road Function.

[16]We follow an analysis by Mitchell [MITC96b].

200 runs	GA	SAHC	NAHC	RMHC
Mean	61,334	>256,000	>256,000	6,179
Median	54,208	>256,000	>256,000	5,775

It is a remarkable fact that for the *Royal Road* function, the GA has been so strikingly "defeated" by the Random Mutation Hill Climber. The above results shed light on two important issues that have been already mentioned:

a) Deception.
b) The Building Block Hypothesis.

We now discuss the causes for the GA's poor performance and analyze the behavior of the hill climber. We propose a special kind of GA called an *idealized GA (IGA)* in which the essence of a GA is typified. By trying to capture the essence of a GA (in an IGA) we shall be able to conclude as to when a GA will be expected to perform optimally.

4.7.2 Spurious Correlation in Genetic Algorithms

One of the reasons why the GA performed badly on the RR function was "hitchhiking" (also referred to as *spurious correlation*): once an instance of a higher-order schema is discovered, its high fitness allows the schema to spread quickly in the population, with zeros in other positions in the string hitchhiking along with the ones in the schema's defined positions.

This slows the discovery of schemas in the other positions, especially those that are close to the highly fit schema's defined positions.

Hitchhiking seriously limits the implicit parallelism of the GA by restricting the schemas sampled at certain loci. Hitchhiking in GA's has also been called "spurious correlation".

It is, therefore, important to realize that a poor behavior of the GA's is not necessarily related to deception. There are other causes that may interfere with its performance.

4.8 Markov Chain Models

In the preceding sections we made an analysis which basically relies on the concept of a *schema*. We were able to arrive at certain interesting conclusions about the SGA. Here we approach the behavior of GAs by modeling them as finite Markov chains. Our treatment follows the one by Rudolph [RUDO97].

Markov chains are stochastic processes in which the probability that the process will

be in state j at time t depends only on the state i at time t-1[17]. A "state" of a finite-population GA is simply a particular finite population. The set of all states is the set of all possible populations of size n. These can be enumerated in some canonical order and indexed by i. We may represent the i-th such population as a vector $\vec{p}$ of length 2^l. The yth element of $\vec{p}$ is the number of occurrences of string y in population P_i. Under an SGA the current population P_j depends only on the population at the previous generation. Therefore, the GA may be modelled as a Markov chain.

The set of all possible populations of size n can be represented by a matrix Ξ in which the columns are all possible population vectors $\vec{p}_i$. There are

$$N = \binom{n + 2^l - 1}{2^l - 1} \tag{4.8.a}$$

populations of size n.

EXAMPLE E.4.8.1

Let $n = 2$ and $l = 2$. The possible populations are

$$P_0 = \begin{Bmatrix} 00 \\ 00 \end{Bmatrix} \; P_1 = \begin{Bmatrix} 00 \\ 01 \end{Bmatrix} \; P_2 = \begin{Bmatrix} 00 \\ 10 \end{Bmatrix} \; P_3 = \begin{Bmatrix} 00 \\ 11 \end{Bmatrix} \; P_4 = \begin{Bmatrix} 01 \\ 01 \end{Bmatrix}$$
$$P_5 = \begin{Bmatrix} 01 \\ 10 \end{Bmatrix} \; P_6 = \begin{Bmatrix} 01 \\ 11 \end{Bmatrix} \; P_7 = \begin{Bmatrix} 10 \\ 10 \end{Bmatrix} \; P_8 = \begin{Bmatrix} 10 \\ 11 \end{Bmatrix} \; P_9 = \begin{Bmatrix} 11 \\ 11 \end{Bmatrix}$$

The matrix Ξ is

$$\Xi = \begin{pmatrix} 2 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 2 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 2 \end{pmatrix}$$

A state for the Markov chain corresponds to a column of Ξ. Now we can set up a transition matrix **P**. **P** is an N × N matrix, and each element $P_{i,j}$ is the probability that population P_j will be produced from population P_i under an SGA.

A finite Markov chain describes a probabilistic trajectory over a finite space S of cardinality $|S| = n$, where, as remarked, the states may be numbered from 1 to n. The probability $p_{ij}(t)$ of passing from state $i \in S$ to state $j \in S$ at step t is called the *transition probability* from i to j at step t. If the transition probabilities are independent from t, i.e., $p_{ij}(t) = p_{ij}(s)$ for all $i, j \in S$ and for all $s, t \in \mathbb{N}$, the Markov chain is said to be homogeneous.

The transition probabilities of a homogeneous finite Markov chain can be gathered

[17]See section 2.6.

in the *transition matrix* $\mathbf{P} = (p_{ij})$. If for each entry, $p_{ij} \in [0, 1]$ and $\sum_{j=1}^{|S|} p_{ij} = 1$ for all $i \in S$ a matrix is called *stochastic*. Given an initial configuration $\vec{p}^{0}$ as a row vector, the configuration of the chain after the *t*-th step is determined by $\vec{p}^{t} = \vec{p}^{0}\,\mathbf{P}^{t}$. Therefore, a homogeneous finite Markov chain is completely determined by the pair $(\vec{p}^{0}\ \mathbf{P})$.

4.8.1 Markov Chain Analysis of Genetic Algorithms

The SGA can be described as a Markov chain. The state of the SGA depends only on the genes of the individuals so that the state space is $S = I\!B^{N} = I\!B^{l \cdot n}$, where *n* denotes the population size and *l* is the number of genes. Each element of the state space can be regarded as an integer number in binary representation. The projection $\pi_k(i)$ represents the segment of length *l* from the binary representation of individual *k* in state *i* and is used to identify single individuals of the population. The probabilistic changes of the genes within the population caused by the genetic operators[18] are captured by the transition matrix **P**, which can be decomposed into a product of stochastic matrices $\mathbf{P} = \mathbf{C} \cdot \mathbf{M} \cdot \mathbf{S}$, where **C**, **M** and **S** describe the intermediate transitions caused by crossover, mutation and selection, respectively. This leads to:

THEOREM 4.8.1.1

The transition matrix of the SGA with mutation probability $p_m \in (0,1)$ is regular.

Proof:

The crossover operator may be regarded as a random total function whose domain and range are *S*, therefore, each state of *S* is mapped probabilistically to another state. Hence, **C** is stochastic. The same holds for other operators and their transition matrices.

The mutation operator is applied independently to each bit in the population and the probability that state *i* becomes state *j* after mutation is expressed by $m_{ij} = p_m^{H_{ij}}(1 - p_m)^{N - H_{ij}} > 0$ for all $i, j \in S$ where H_{ij} denotes the Hamming distance between the binary representation of state *i* and state *j*. Thus **M** is positive.

The probability that selection does not alter the state generated by mutation can be bounded by

[18] Crossover, mutation and selection.

$$s_{ii} \geq \frac{\prod_{k=1}^{n} f(\pi_k(i))}{\left(\sum_{k=1}^{n} f(\pi_k(i))\right)^n} > 0 \qquad \text{(4.8.1.a)}$$

so that **S** is c-allowable. From lemma 2.6.1. we know that **P** = **C** · **M** · **S** is positive. Since every positive matrix is regular the proof is completed. □

COROLLARY 4.8.1.2

The SGA as defined in theorem 4.8.1.1. is an *ergodic* Markov chain, that is, there exists a unique limit distribution for the states of the chain with nonzero probability to be in any state at any time regardless of the initial distribution.

This implies that the initial distribution $\vec{p}^{\,0}$ has no effect on the limit behavior of the Markov chain. The initialization of the algorithm can be done arbitrarily.

DEFINITION 4.8.1.3

Let $Z_t = \max\{f(\pi_k^{(t)}(i)) \mid k = 1,\ldots,n\}$ be a sequence of random variables representing the best fitness within a population represented by state *i* at step *t*. An SGA converges to the global optimum iff

$$\lim_{t\to\infty} P\{Z_t = f^*\} = 1 \qquad \text{(4.8.1.b)}$$

where $f^* = \max\{f(b) \mid b \in I\!B^l\}$ is the global optimum of the problem.

This leads to:

THEOREM 4.8.1.4

The Simple Genetic Algorithm does not converge to a global optimum.

Proof:

Let $i \in S$ be any state with $\max\{f(\pi_k(i)) \mid k = 1,\ldots,n\} < f^*$ and $p_i^{\,t}$ the probability that the GA is in such state *i* at step *t*.

Clearly $P\{Z_t \neq f^*\} \geq p_i^{\,t} \Leftrightarrow P\{Z_t = f^*\} \leq 1 - p_i^{\,t}$. From theorem 2.6.1.3. the probability that the GA that is in state *i* converges to $p_i^{\infty} > 0$. Consequently,

$$\lim_{t\to\infty} P\{Z_t = f^*\} \leq 1 - p_i^{\infty} < 1 \qquad (4.8.1.c)$$

so that condition (4.8.1.b) is not fulfilled. □

When working with GAs the logical practice is to keep track of the best individual, so that one might argue that Markov chains do not represent a practical GA. In fact, after a finite number of transitions the global solution will be visited and copied. This is consequence of the following:

THEOREM 4.8.1.5

In an ergodic Markov chain the expected transition time between an initial state *i* and any other state *j* is finite regardless of the states *i* and *j*. □

In section 4.7.5., conclusion 2 states that to approach an IGA the desired schemas must be sequestered. This is the principle behind elitist models. This intuitive conclusion is formalized in the following

THEOREM 4.8.1.6

The SGA maintaining the best solution found over time *after* selection converges to the global optimum. □

Proof.

Let us adapt the Markov chain description by enlarging the population by an additional, say, "super individual" which does not take part in the evolutionary process. The cardinality of the space grows from $2^{n\cdot l}$ to $2^{(n-1)\cdot l}$. For convenience let the super individual be placed at the leftmost position in the $(n+1)$-tuple and let it be accessible by $\pi_0(i)$ from a population at state *i*. The transition probabilities of those states containing the same super individual string are assumed to be listed one below the other in the transition matrix and the better the super individual's fitness the higher the position of the corresponding state in the matrix. Since the super individual's string is not affected by a genetic operator, the extended transition matrices for crossover, mutation and selection ($\mathbf{C}^+$, $\mathbf{M}^+$, $\mathbf{S}^+$) can be written as block diagonal matrices:

$$C^{+} = \begin{pmatrix} C & & & \\ & C & & \\ & & \ldots & \\ & & & C \end{pmatrix},\ M^{+} = \begin{pmatrix} M & & & \\ & M & & \\ & & \ldots & \\ & & & M \end{pmatrix},\ S^{+} = \begin{pmatrix} S & & & \\ & S & & \\ & & \ldots & \\ & & & S \end{pmatrix} \qquad (4.8.1.d)$$

with 2^l square matrices **C**, **M** and **S** of size $2^{nl} \times 2^{nl}$ possessing the same structure as in the ergodic case so that

$$C^{\infty} M^{\infty} S^{\infty} = \begin{pmatrix} CMS & & & \\ & CMS & & \\ & & \ddots & \\ & & & CMS \end{pmatrix} \qquad (4.8.1.e)$$

CMS > **0**. The copy operation is represented by an *upgrade* matrix **U** which upgrades an intermediate state containing an individual better than its super individual to a state where the super individual equals the better individual. Let *b* denote the best individual of the population at any state *i* excluding the super individual. Then $u_{ij} = 1$ if $f(\pi_0(i)) < f(b)$ with $j \triangleq (b, \pi_1(i), \pi_2(i), \ldots, \pi_n(i)) \in S$, otherwise $u_{ii} = 1$.

Thus, there is exactly one entry in each row which does not hold for the columns because for every state $j \in S$ with $f(\pi_0(j)) < \max\{f(\pi_k(j)) | k = 1, \ldots, n\}$ one gets $u_{ij} = 0$ for all $i \in S$. In other words, a state either becomes upgraded or remains unaltered. Therefore, the upgrade matrix can be written as

$$U = \begin{pmatrix} U_{11} & & & \\ U_{21} & U_{22} & & \\ & & \ddots & \\ U_{2^l,1} & U_{2^l,2} & \ldots & U_{2^l,2^l} \end{pmatrix} \qquad (4.8.1.f)$$

with submatrices $\mathbf{U}_{ab}$ of size $2^{nl} \times 2^{nl}$. For convenience, let us assume that the problem has only one global optimizer. Then only $\mathbf{U}_{11}$ is a unit matrix whereas all matrices $\mathbf{U}_{aa}$ with $a \geq 2$ are unit matrices with some zero diagonal entries. With **P** = **CMS** the transition matrix for the GA becomes

$$P^{\infty} = \begin{pmatrix} P & & & \\ & P & & \\ & & \ddots & \\ & & & P \end{pmatrix} \begin{pmatrix} U_{11} & & & \\ U_{21} & U_{22} & & \\ & & \ddots & \\ U_{2^l,1} & U_{2^l,2} & \ldots & U_{2^l,2^l} \end{pmatrix}$$

$$P^{\infty} = \begin{pmatrix} PU_{11} & & & \\ PU_{21} & PU_{22} & & \\ & & \ddots & \\ PU_{2^l,1} & PU_{2^l,2} & \ldots & PU_{2^l,2^l} \end{pmatrix} \qquad (4.8.1.g)$$

with $\mathbf{PU}_{11} = \mathbf{P} > \mathbf{0}$. The submatrices $\mathbf{PU}_{a1}$ with $a \geq 2$ may be gathered in a rectangular matrix $\boldsymbol{R} \neq \mathbf{0}$ so that theorem 2.6.1.4. applies from which the corresponding GA converges to a global optimum. □

We state a complementary theorem whose proof is an adaptation of the above:

THEOREM 4.8.1.7

The SGA maintaining the best solution found over time *before* selection converges to a global optimum. □

Note that theorems 4.8.1.6. and 4.8.1.7. do not cover the case of elitist selection. When using elitist selection the best individual is not only maintained but also utilized to generate new individuals. This algorithm has another transition matrix and, therefore, different search dynamics which may be better in some cases and worse in other cases. Clearly, however, it converges to a global optimum.

In the case of a canonical SGA global convergence is not guaranteed. The reason is quite clear: for an SGA there is a minimal probability bounded from zero to lose the global optimum solution at each generation. It follows from the Borel-Cantelli Lemma[19] that this event will occur with probability one. On the other hand, there is a minimal probability to find again a global solution if it was lost, so that this event will also occur with probability one. In fact, the global best solution will be lost and found infinitely often so that the sequence of best individuals found over time in the SGA is an irreducible Markov chain on the state space {0, 1, ..., *n*} which does not converge although the expectation does.

4.8.2 Conclusions

The previous analysis shows that global convergence is not an inherent property of an SGA. In other words, the original SGA cannot be regarded as an optimization algorithm for the static optimization problems because it is provable that it will not converge to any subset of states containing at least one global solution, even in infinite time. Static optimization, however, was not the original purpose in the design of the SGA. In fact, the interest was concentrated on a strategy which performs an optimal allocation of trials so as to minimize expected losses in an uncertain environment, possibly with time varying rewards. The *fundamental theorem of GAs* (equation 4.3.a) does not imply that the SGA will converge to the global optimum in static optimization problems. Moreover, due to the irreducibility of the SGA it is clear that the SGA will not converge at all.

It also shows, however, that by the simple expedient of maintaining the best observed individual we may guarantee such global convergence. This is a very strong result which gives solid foundation to the claim made in our opening paragraphs that GAs are a robust and general optimization alternative.

---

[19]See W.Feller, *An Introduction to Probability Theory and Its Applications, Vol. 1.*, Wiley, 1970.

Therefore we have now two proven facts:

a) GAs are near optimal strategies to attack *dynamic* problems.
b) GAs are guaranteed to converge to a global optimum in *static* problems.

As also pointed out, however, certain optimizing techniques, such as hill climbers, may outperform GAs in terms of how efficiently they find the said static solutions. We would now like to discuss certain heuristic rules which will allow us to increase the GAs' efficiency.

4.9 Non-Standard Models

At this point, it should be clear that we have at least two broad fields of application for GAs: *dynamic* optimization problems and *static* optimization problems. It has also been proved that either of these fields may be approached with success with the proper GA variation. Our concern is now to find a way to make these effective techniques more efficient. By modifying the basic strategy of the SGA we come to *non-standard models of Genetic Algorithms*.

4.9.1 Elitism

As discussed above, when one does keep a copy of the best individual we may guarantee global convergence for an optimization problem. There are variations in elitist models which we have denoted as *partial elitism* and *full elitism*. By partial elitism we mean that in a population of size n we keep a copy of the $\tau < n$ best individuals up to generation k. In the case discussed in theorems 4.8.1.6. and 4.8.1.7. $\tau = 1$. By full elitism we mean that we keep a copy of the best n individuals up to generation k. In other words, given that we have tested nk individuals by generation k, our population will consist of the best n up to that point.

In figure F.4.9.1.1. we depict the case of full elitism. Notice that in generation k we have selected the best possible n individuals out of the nk total individuals considered up to that point. In so doing we artificially force the population to lean towards a focalized set of hyperplanes. The obvious risk is that we restrict the search space in a way that hampers the GA excessively. To avoid this risk we may alter the multi-faced die strategy as discussed in what follows.

4.9.2 Deterministic Selection

In the SGA we have considered proportional selection as per equation 1.1.1.c. In fact, the schema theorem has been derived from such consideration. Now we consider the *deterministic* selection outlined in section 4.1.1.3. In deterministic selection we no longer

rely on the individual fitness to determine the most desirable descendants. Rather, we propose to emphasize genetic variety by imposing a strategy which enforces crossover of predefined individuals. There are two contrasting points of view. In one of these, we encourage the genes of the best individuals to cross between themselves; in the other we encourage the best individuals to cross with the *worst* ones. One of these two strategies is called the Nietzsche model (NM), where the best elements of the population intermix in an effort to preserve the "best" genes. The other strategy is called the Vasconcelos model (VM), where the "best" individuals are intermixed with the "worst" individuals in an effort to explore the widest range of the solution space.

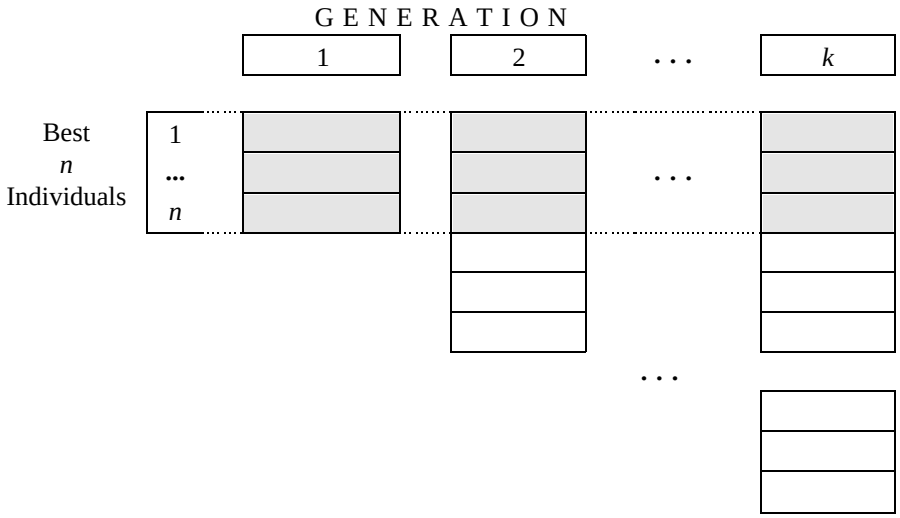

Figure F.4.9.1.1. Full Elitism.

4.9.2.1 Nietzsche Model

In this model we adopt the strategy of deterministically selecting individual *i* to cross it with individual *i+1*. It assumes that full elitism has been applied. Therefore, we are forcing the genes of individuals 1 and 2 to cross; the genes individuals 3 and 4 to cross; and so on up to the genes of individuals *n-1* and *n* to cross.

A superficial analysis would suggest that a mechanism which enhances the beneficial traits of the individuals should lead to a better algorithm. However, simulation of the process does not enforce this view. The reason is, simply, that to sequester the desirable schemas in this fashion leads to what is called *premature convergence*. That is, the deterministic selection of best individuals in this fashion does not allow for a sufficiently vast exploration of the solution space. In a sense, we are curtailing the exploration phase while maximizing the exploitation of positive features of the individuals.

4.9.2.2 Vasconcelos Model

In this model we adopt the strategy of deterministically selecting individual *i* to cross it with individual *n-i+1*. As in the NM, we assume full elitism. Hence, here we adopt a strategy which, superficially, destroys the good traits of the best individual by deliberately crossing it with the worst individual. However, when taken in conjunction with full elitism this strategy leads to the implicit analysis of a wider variety of schemas (i.e. it maximizes the exploration of the solution landscape). The exploration of such vast landscape is focused via the full elitism implicit in the model.

The VM allows the algorithm to trap the best schemas without restricting the search space in a sensible way. In this fashion we are able to approximate the IGA of section 4.7.4. We now make a brief review of the demands for any GA to approach an IGA:

a) No single locus is fixed at a single value in a large majority of the strings of the population.

This is clearly achieved because we are deterministically disrupting the troublesome locus' by best-worst crossover.

b) Selection has to be strong enough to preserve desired schemas that have been discovered but also should prevent significant hitchhiking on some highly fit schemas.

Because we are working with full elitism we preserve desired schemas and, as before, we avoid hitchhiking by crossing over dissimilar individuals.

c) The crossover rate has to be such that the time for crossover that combines two desired schemas is small with respect to the discovery time for such schemas.

Full elitism guarantees that, even in the Vasconcelos strategy, the worst individual for population *k* is in the top $1/k$ percentage. For example, if $n = 50$ and we look at the 20th population, the individuals in such population are among the best 5% of the total individuals analyzed. This characteristic is incrementally exposed as the GA proceeds.

d) The string has to be long enough so that the speedup factor *N* (in equation (4.7.4.c) is significant.

This is the only element we cannot guarantee. It seems that for relatively small *N*'s hill climbers may outperform our Genetic Algorithms.

4.10 Self-Adaptation

When running a GA there are several parameters that are to be set *a priori*: Three of these are the most common: *a)* The crossover rate (P_c), *b)* The mutation rate (P_m), and *c)* The size of the population (N).

In many cases the user tries to fine tune these parameters by making a number of runs on different "representative" case problems (see, for instance De Jong[DEJO75], Galavíz [GALA96], pp. 156]). Here we discuss in more detail the *self-adaptation* of the mentioned parameters, which was already briefly touched in section 4.1.1.12. under the endogenous variation.

In a self-adaptive GA the three parameters are included as an extension of the genome in such a way that the parameters evolve along with the individual. This was also mentioned when discussing Evolution Strategies, in section 3.1. The idea behind self-adaptation is that not only does the GA explore the solution landscape but the *parameter landscape* as well. In this way, the genome is divided in two sub-genomes: a *strategy genome* and a *solution genome*. In terms of a classical GA, the solution genome corresponds to what has simply been referred, up to this point, as the *genome*. Both sub-genomes are subject to the genetic operators. We should consider the parameter sub-genome as a set of three sub-genomes which are functionally independent. The self-adaptive genome is shown in figure F.4.10.1. Notice that the size of the population *N* is implicitly dealt with by considering not the population's size itself, but rather the number of descendants product of crossover. Recall, for instance, that in the Royal Roads the number of descendants was handled via sigma truncation. There we had an embryo self-adaptation strategy, for the parameter determining the details of offspring size was set a priori.

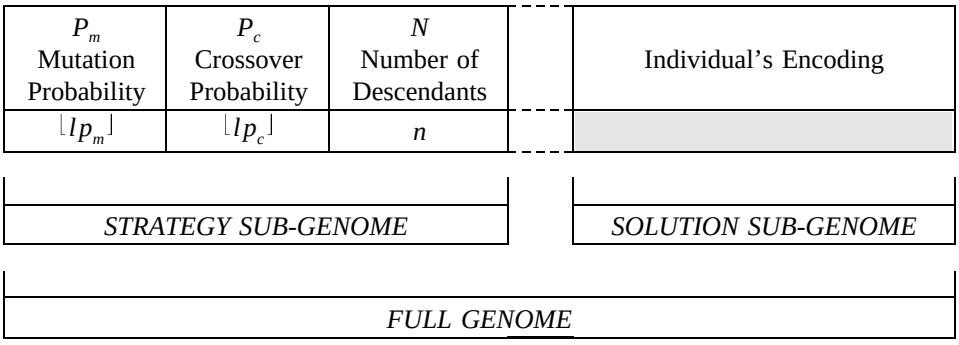

Figure F.4.10.1. Self-Adaptive Genome

4.10.1 Individual Self-Adaptive Genetic Algorithm

In this case, the different parameters take their values from considerations affecting the individuals as independent units, as opposed to individuals as members of the population (see next section). For example, we may encode the *control parameters* as follows:

a) *Probability of Mutation*

Let $p_m \in [0,1]$ be the probability of a bit mutation for a given individual. p_m is represented in binary as an integer n where $n = \lfloor Kp_m \rfloor$ and $K \in \mathbb{I}$. There are N mutation rates which we denote by $(p_m)_i$ $i = 1,2,...,N$. These evolve in time under the GA's operators. Each individual is then mutated with probability $(p_m)_i$.

b) *Probability of Crossover*

Let $p_c \in [0,1]$ be the probability of crossover for a given individual. As in the case of p_m, p_c is encoded in binary as an integer n where $n = \lfloor Kp_c \rfloor$. The code of p_c is also a genetic string and is subject to the genetic operators. Given two individuals with crossover probabilities p_{c1} and p_{c2}, they are crossed with probability:

$$p_{c12} = \frac{p_{c1} + p_{c2}}{2} \tag{4.10.1.a}$$

c) *Number of Descendants*

Each individual has a number of descendants which is also encoded and considered to be a genetic string subject to the genetic operators. Let c_1 and c_2 be the number of descendants of two individuals. When such individuals are crossed the offspring they will generate is given by:

$$n = \left(\frac{c_1 + c_2}{2} \right) \left(\frac{\overline{f_{1,2}}}{f_{\max}} \right) \tag{4.10.1.b}$$

where $\overline{f_{1,2}}$ is the mean fitness for these two individuals and $f_{\max}$ is the best fitness for the population in the current generation. The number of descendants determined as described insures that poor genetic structures have less chance to survive than the better ones.

4.10.2 Population Self-Adaptive Genetic Algorithm

In this case of self-adaptation the way the operators affect the performance of the GA takes into consideration the population (for any given generation) as a whole. In a sense, this self-adaptive GA is aimed at improving the mean values of the population rather than

the values of each individual.

a) *Probability of Mutation*

As in the individual self-adaptive scheme, p_m is encoded in every individual. Here, however, the mutation rate for the whole population in the k-th generation g_k is calculated as follows:

$$(p_m)_k = \frac{1}{N} \sum_{i=1}^{N} (p_m)_i \qquad (4.10.2.a)$$

Therefore, the mutation operator's rate is fixed for all the individuals during g_k.

b) *Probability of Crossover*

p_c is, as before, encoded in the genome of every individual. Now, in generation g_k the crossover rate is given by

$$(p_c)_k = \frac{1}{N} \sum_{i=1}^{N} (p_c)_i \qquad (4.10.2.b)$$

Here, again, the crossover operator's rate is fixed for all the individuals during g_k.

c) *Number of Descendants*

As before, n, the number of descendants, is encoded in the genome of every individual and, as before, the number of descendants n_k in the k-th generation is given by

$$n_k = \frac{1}{N} \sum_{i=1}^{N} n_i \qquad (4.10.2.c)$$

As in the two preceding cases, the number of descendants is fixed for all the individuals during g_k.
□

Utilizing a self-adaptive strategy one is freed from offset arbitrary parameter selection, to a certain extent. It is usual to set upper bounds on the possible values encoded in p_m, p_c and n. In that sense, there is still an arbitrary selection of initial parameter values. However, individuals which represent the better parameters for the particular problem are allowed to *learn* from the problem that is being solved. The self-adaptive alternative has been shown to compare favorably [GALA96, p. 162] with traditional fixed parameter GAs.

4.11 An Eclectic Universal Genetic Algorithm

At this point we have all the elements, both theoretical an practical to propose what we shall call an *Eclectic Universal Genetic Algorithm (UGA)*. Eclecticism here refers to the fact that we are willing to adopt the strategies we consider best regardless of the problem. In fact, we arrive at a mixed algorithm: strictly not merely a GA any more. Universal is meant to stress the fact that the variation of the GA to be discussed is applicable to a wide range of problems without the need for special considerations. In fact, we shall utilize this UGA to solve problems both numerical and non-numerical. We shall apply them both to optimization and to learning. This will be accomplished without changing the EGA and simply by designing the fitness function in a way exactly analogous to the way it was described in chapter 3.

Throughout the text we have repeatedly stressed the fact that GAs are flexible, easy to adjust to the problem in hand, robust and able to perform global search characteristics in the solution landscape. We also showed that they approach a near optimal tool for dynamic problems and guaranteed to converge in static problems (via elitism).

The disadvantage that is frequently cited is that GAs, as all artificial intelligence "weak" methods, incorporate the knowledge about the problem in the structure of the method itself, rather than explicitly displaying it in the strategies involved. In the GA this problem dependent "knowledge" is represented by the values that must be set in its operators and the values that these operators take.

We have given theoretical arguments that explain the shortcomings of GAs and, consequently, allow us to take the corresponding counter-measures so that we may overcome them. Via the IGA we were able to determine the desirable features an effective GA must face to avoid deception and spurious correlation.

Even when we are able to determine the causes for possible GA's suboptimal performance we are still faced with the parameter setting problem. This may be overcome by self-adaptation. In what follows we make a brief synthesis of these matters and add a hybridizing element in the GA which will allow us to overcome the N-factor problem discussed in the last paragraph of section 4.9.2.2.

4.11.1 Selection, Elitism and Self-Adaptation

The EGA includes the following components:

a) Deterministic selection.
b) Elitism.
c) Self-Adaptation.

We have already discussed at length the possibilities and advantages of each of these elements in the preceding sections. It only remains to define the particular variation of each component.

Selection is done as per *Vasconcelos' strategy*: cross individual i with individual $N-i+1$ for $i = 1, 2, ..., N$.

Elitism is *full* retaining always the best N individuals out of the Nk tested up to

generation *k*.

Self-Adaptation is on the *population*. Three parameters are encoded and self-adapted: *mutation*, *crossover* and *number of descendants*.

Mutation is uniform.

Crossover is *annular*.

In annular crossover, the genome (as shown in figure F.4.11.1.1.) is no longer seen as a linear collection of bits but rather as a ring whose leftmost bit is contiguous to its rightmost bit.

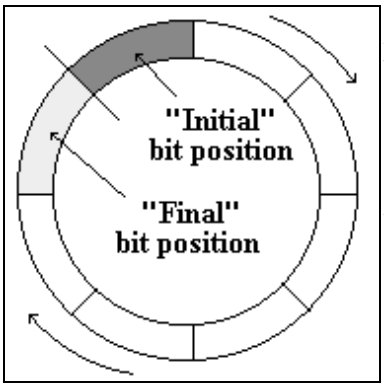

Figure 4.11.1.1. Annular Genome.

When applying annular crossover, there are two parameters to consider for each interchange:

a) The starting crossover locus.

That is, where the segment to be extracted from the individual starts.

b) The length of the semi-ring.

That is, how many genes of the individual are to extracted.

Clearly, for a genome of length *l* there are *l* possible locus and *l*-1 possible lengths. When extracting the first individual's genes, however, we must no longer concern ourselves with position encoding dependencies such as the ones discussed in section 4.1.1.10. An example of annular crossover is shown in figure F.4.11.1.2.

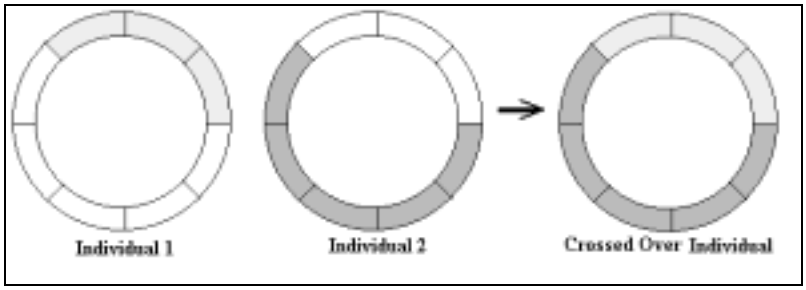

Figure 4.11.1.2. Annular Crossover.

As already mentioned, this algorithm approaches the behavior of an idealized Genetic Algorithm. In including a self-adaptive behavior it modifies its behavior without

impairing the desirable characteristics.

4.11.2 Adaptive Hill Climbing

As exposed by the experiments summarized in table T.4.7.1.1., the Random Mutation Hill Climber (HC) is capable of outperforming the SGA in certain functions. To take advantage of the hill climber in cases as this one we include a RMHC as part of the algorithm. That is, the EGA consists of a self-adaptive GA *plus* an HC. How do we determine when the HC should be activated instead of the GA proper? We do this with a self-adaptive mechanism which we describe next.

a) The first step is to define two bounds:
   1) The minimum Hill Climber percentage (η_λ).
   2) The maximum Hill Climber percentage (η_μ).

where

$$0 < \eta_\lambda < \eta_\mu \leq 1 \tag{4.11.2.a}$$

The HC algorithm will then be activated, at least, η_λ of the time and, at most, η_μ of the time.

b) As a second step, we must define two other bounds:
   1) The minimum number of function evaluations to be performed by the HC upon invocation (N_λ).
   2) The maximum number of such function evaluations (N_μ).

These two bounds are given as a percentage of the population's size *N*. The actual percentage of HC function evaluations (relative to *N*) is included in the strategy sub-genome and is subject to the genetic operators. The actual number of evaluations upon HC invocation N_η in generation *k* is given by

$$(N_\eta)_k = \frac{1}{N} \sum_{i=1}^{N} (N_\eta)_i \tag{4.11.2.b}$$

c) At generation g_k the hill climber's effectiveness is evaluated from

$$\eta_\phi = \frac{1}{N} \sum_{i=1}^{N} (\iota_\eta)_i \tag{4.11.2.c}$$

where

$$\iota_\eta = \begin{cases} 1 & \textit{if individual was found by the HC} \\ 0 & \textit{otherwise} \end{cases} \tag{4.11.2.d}$$

To be able to determine ι_η's value we must include a new element in the genome. This element is of type boolean. It will be set when the individual has been found by the HC and reset otherwise.

The genome for the EGA is shown in figure F.4.11.2.1. In it we may find all the elements for the self-adaptive scheme *and* the HC algorithm.

P_m Mutation Probability	P_c Crossover Probability	N Number of Descendants	ι_η Originated by HC	HC Function Evaluations	Individual's Encoding
$\lfloor lp_m \rfloor$	$\lfloor lp_c \rfloor$	n	η	N_η	

STRATEGY	*SOLUTION*

Figure F.4.11.2.1. Self-Adaptive Genome for Eclectic Genetic Algorithm.

d) Denoting the probability of invoking the HC by η_τ, we have:

$$\eta_\tau = \begin{cases} \eta_\lambda & \text{if } \eta_\phi < \eta_\lambda \\ \eta_\phi & \text{if } \eta_\lambda \le \eta_\phi \le \eta_\mu \\ \eta_\mu & \text{if } \eta_\phi > \eta_\mu \end{cases} \qquad (4.11.2.e)$$

e) Generate a random number ρ_K, where $0 < \rho_K \le 1$. Prepare to invoke the HC if $\eta_\tau \le \rho_K$.

f) Once the HC is scheduled to start, the string upon which it will operate is selected randomly from the first five in the population. Recall that the individuals in the population are ordered from best (individual 1) to worst (individual N). Therefore, to select the string ι_η upon for the HC to operate, we make

$$\iota_\eta = \lfloor R \times 5 \rfloor + 1 \qquad (4.11.2.f)$$

where R is a random number uniformly distributed and $0 < R \le 1$. □

The result of the strategy just outlined is to guarantee that the HC will be active whenever it has proved to be effective. The probability that the HC will override the GA proper is, however, bounded above by η_μ. This prevents the HC from taking over the whole process. On the other hand, the HC will be invoked with probability $\ge \eta_\lambda$ which, in turn, avoids the possibility that due to poor performance of the HC at some generation g_k, the HC is shut out for the rest of the process. In essence, therefore, the EGA

incorporates an HC process which is self-adaptive on two accounts: a) Because its activity is determined by its effectiveness, and b) Because the adequate number of function evaluations is evolved as the GA unfolds.

It was remarked earlier that the name "Hill Climber" refers to the fact that these processes are thought to zeroin on optimality points when the algorithm has reached a neighboring space in the solution landscape. Here, however, the HC serves a double purpose: a) It does indeed zeroin on local, close maxima, and b) It enforces population variety by exploring new schemas which the GA would otherwise pass by. The most striking fact about this mixed mode (GA-HC) algorithm is that it is the GA which actually does the fine search of local optima, with the HC serving as a triggering agent which locates suboptimal solutions quite efficiently. The reader will have the opportunity to verify the EGA's behavior with the software included in the companion diskette.

Notice that, in a GA as the one just described, the Schema Theorem (equation 4.3.a) is clearly not applicable. In fact, seldom do the criteria employed there have any bearing upon the way the EGA may be analyzed. As stated before, the theoretical treatment of genetic algorithms is still in its infancy and the generalization of the basic concepts derived from schema or Markov chain analysis is still lacking. This is not to say that we may not benefit from the theory already developed. It does give us an efficient way to establish comparisons between different GA variations. Therefore it will be useful to establish a relative performance criterion to evaluate the eclectic genetic algorithm and other variations which will be explored later in the text.

The EGA may be characterized, according to the proposed taxonomy, as:

$$EGA = \Gamma(1, 1, 3, 2, 2, 3, 1, 3, 1, 1, 1, 2, 1) \qquad (4.11.2.g)$$

Although the schema theorem is away from the EGA's spirit we may still remark, if only qualitatively, how it does compare with the *IGA* whose desirable features were identified from a traditional schema analysis. By incorporating self-adaptation and hill climbing the eclectic genetic algorithm approaches the behavior of the *IGA* discussed above since:

a) Via Vasconcelos' strategy no locus is fixed at a single value.
b) Via full elitism, desired schemas are sequestered to preserve good traits.
c) Via self-adaptation crossover is chosen so as to dynamically maintain the best crossover rate.
d) The speedup factor (in equation 4.7.4.c) is at least equal to the hill climber's due to the fact that a hill climbing mechanism is explicitly included.