A Genetic Algorithm White Paper

An Introduction to Genetic Algorithm Implementation,
Theory, Application, History and Future Potential.

By: Salvatore R. Mangano

Contents:

I. Introduction

Genetic algorithm (GA) technology has been around for over 30 years. However, applications based on this technology are just recently (past 5 years) being fielded in earnest. The purpose of this white paper is to give managers, consultants, analysts and other interested individuals the background necessary to assess the viability of genetic algorithm technology towards their problems and applications.

This paper is organized such that it can be read as a whole or in parts without loss of continuity. I have compartmentalized the information into sections that address specific aspects of the GA field. Individuals with no knowledge of GA's will certainly want to read sections II and III. Others with only a passing familiarity with GA's may be surprised at the myriad applications that are finding GA solutions. These individuals should certainly not pass up section VI. Those who are bent towards theory should consult section V. Others who prefer to see that there is some history behind the technology should read section IV. Finally those individuals who know everything there is to know about GA's can certainly find some insight (and/or possibly some amusement) in section VII.

The tone of this paper is purposefully lite and in the first person. I wished to create a more entertaining and conversational piece than is normally found in your average "white paper". This is not meant to imply that I do not take the subject matter seriously. The 14 hour days that I have been dedicating to writing, programming, consulting and researching GA’s (as well as other related technology) over the past 2 1/2 years certainly attests to my seriousness. Although the information is sometimes presented here in a colorful manner, I have gone to great lengths to keep the information accurate. When I state opinion, I clearly say so. When I state fact, you can be assured that it was obtained from one of the books listed in the bibliography. I do not quote references every few sentences like most academic works do because this interrupts the flow. I instead give a brief synopsis of the kinds of material that can be found in the references I used under the titles in the bibliography. This will allow you to explore topics in more depth if you so desire.

II. Hi, My Name is GA. G Who?!?!

When people I meet either professionally and socially ask me what I am working on these days, I cringe because I know an honest answer will usually result in a puzzled response. "Genetic Algorithms, Huh?, I thought you were a computer consultant!" Unfortunately, many of these types of responses come from people who are otherwise knowledgeable about the latest computer technology. They have heard of object oriented programming, graphical user interfaces, CASE and often even neural networks. But genetic algorithms (GA’s) usually cause amazed or incredulous expressions. The problem is exacerbated by the fact that a short and sweet explanation of GA’s will often give the person a better idea of what I am doing, but leave them suspect that I have gone off the deep end. I am sure they think that I have taken a path that is sure to ruin my otherwise successful consulting career.

The purpose of this section is to explain what GA’s are in a way that hopefully will leave the reader (and my friends and colleagues) with a much better impression of the technology. This technology is powerful. It certainly deserves as much respect as other mainstream technologies like OOP, CASE or neural nets. If I am successful here then you will agree. If you don’t, then it’s a reflection on my limits as a communicator and not on the field of genetic algorithms!

We will start out with a simple definition of Genetic Algorithms:

A GA is a process which mimics the way biological evolution works. Like evolution, it operates on a population of individuals which represent potential solutions to a given problem. It then seeks to produce better (more fit) individuals (solutions) by combining the better of the existing ones (breeding). Using a "survival of the fittest tactic" it weeds out the bad and tends to produce more of the good individuals. Not only does it produce more of the good solutions but better and better solutions. This is because it combines the best traits of parent individuals to produce superior children. This combination operator is called crossover. The term "genetic algorithm" comes from the fact that individuals are represented as strings of bits analogous to chromosomes and genes. In addition to recombination by crossover, we also throw in random mutation of these bit-strings every so often. This keeps the GA from getting stuck at good but non-optimal solutions.

So far our rather simple definition has proceeded in the way that many other simple definitions of new computer technology proceed. We explain the technology in terms of its basis in the natural world. For example, neural networks are usually explained by their basis in brain science (neurobiology) and object oriented technology is explained by the obvious fact that the world is made up of different objects each having their own behavior and inter-relationships.

Why then do people (at least in my personal experience) give more immediate credibility to neural networks and object oriented programming then they do GA’s? I will attempt a series of 5 guesses at the answer to this question. As in all "why" questions there is always more than one answer and each is usually only partially right. The important goal is that in the process of answering this question I hope to provide some more insight into what GA’s are.

Guess #1: The Familiarity Problem

When was the last time you studied evolution? For many people, the answer to this question is probably in college biology or perhaps as far back as high school. Many students studying for a degree in computer science will usually take courses in chemistry and physics, but rarely is biology required. Therefore, although almost every learned person has heard of Darwin, Evolution and the expression "survival of the fittest", they usually only have a very high level and often faulty knowledge of the underlying theory. You just don’t see evolution happening every day! On the other hand, powerful brains and the ubiquitous objects confront us daily. Computers and brains have been compared ever since the first primitive computing machines where invented. For computer programmers, object oriented technology seems just to be the next logical step from the structured "function oriented" programming they are used to. But, Evolution and Computers? -- that sounds weird!

The problem, I think, is that the word "genetic" conjures up images of test tubes, double helixes of molecules and pea plants. The word "evolution" brings to mind apes, fossils and creationism (the competing "science" of the religious right). Putting "genetic" together with "algorithm" or "evolution" with "programming" just does not produce an immediate conceptual click. Sometimes names can be unfortunate. The seminal work on genetic algorithms, *Adaptation in Natural and Artificial Systems* by John Holland doesn’t use this term (it was coined later). Even in the latest edition, the entry for "Genetic Algorithm" in the index refers the reader to "Adaptive Plans". So let’s, for a moment, make believe that the term "genetic algorithm" does not exist. We will instead define and talk about a new computer technology called "Adaptive Oriented Programming" (Are you feeling warm and fuzzy yet?). Here is a explanation of Adaptive Oriented Programming as given in a hypothetical presentation by Man Machine Interfaces to a group of IS managers, analysts and programmers (*complete with editorial commentary, in italics*):

Adaptive Oriented Programming is a new computer methodology (*IS people love this word*) that is useful in any problem which involves optimizing, searching and in some cases learning (*optimizing and searching are good, learning is a risky term but many of these people have heard of Artificial Intelligence (AI) so why not throw it in*). Adaptive Oriented Programming works by starting out with a set (*"sets" are math and math is "good"*) of poor to mediocre solutions to the given problem (*okay, they know they can get a hold of some of these*) and altering them by a standard (*another good word*) set of operations to quickly converge on close to optimal and often optimal solutions. The primary operations that are employed are called **stochastic alteration** and **selective recombination** [1] (*we may have now scared some of our audience because of the complicated sounding terms so we quickly recover by saying...*). These operations are actually not as complicated as their names may imply and in any case they come pre-implemented in the various Genet... (*err, aaahhh, cough, cough, I mean...*) Adaptive Oriented Programming Tool Kits available (*whew, almost said the G word! The IS managers quickly forget this slip anyway as soon as they hear "tool kit" because they all love tools and so does their* *staff*). Adaptive Oriented programming has had great success in many fields such as Operations Research, Engineering and Design, Financial Analysis and even Expert Systems and Neural Networks (*we are sure that at least one of these fields is the apple of some IS'ers eye*).

This talk proceeds to give specific examples of applications and results with lots of graphs and other visual aids. We even give a little more detail on how the tool kits work to avoid giving the impression that the technology is black magic (*black magic is definitely bad!*) All through our talk, however, we are very very careful not to mention the words genetic or evolution (*we use "adaptive"*), population (*we use "set of solutions"*), selection or crossover (*we use "selective recombination"*) and mutation (*we use "stochastic alteration"*). Of course we never, ever ever say or imply random (*because random is very very very bad -- see Guess #5*).

This little bit of fun is not meant to belittle the IS manager. Some of my earliest mentors came from this ilk and they were very intelligent and forward looking individuals. But they are also realists and as realists they know that someone has to pay for every new technology. That someone is usually the least technical person in their organization. So if they don't feel comfortable with the technology, how will they ever sell it? They won't.

So the long and the short of it is that familiarity is good and obscurity is bad. Before GA's are accepted they must become common but to become common they must be accepted - the old catch 22. Time and success will hopefully put GA's in the hands of more companies. After all, I remember when object oriented programming was a four letter word (because it came right after a lot of money was spent on CASE tools embodying older methodologies) and neural networks scared the dickens out of one of my former bosses (and probably still does!).

Guess #2: The Time Problem

Those who don't have the "familiarity problem" usually have the "time problem". These people are well versed in evolutionary theory and know that one of this theory's tenets is that significant evolutionary change occurs over large spans of time. Spans so large, they are known as geological time spans (time spans of major geological changes in the earth) as opposed to human (70 or so years) or even historical (centuries) time. They, therefore, claim that evolution is a poor paradigm for computation because biology and history have shown that nothing major can be achieved in even thousands of years. So certainly seconds, hours or even days of computer time on even the best computers could not produce significant results for algorithms based in evolution.

The mistake in such an argument is that it assumes that large time periods are a necessary prerequisite to adaptive evolutionary processes and not simply just an side effect of natural adaptation. The large time periods are certainly a side effect, since all advanced biological organisms take at least one or two years (and often many more) to reach sexual maturity (ability to reproduce). Add to this the fact that the environment that these individuals are reared in does not usually place a high degree of adaptive pressure on them. By this, I mean that environments usually have a stability measured in tens if not hundreds of years. For example, the mountain goats that lived in the Andes 200 years ago would probably be just as viable as the mountain goats that live there today. Evolutionary change is slow because geological change is slow, not because evolution is an inherently slow process. This can be demonstrated in the laboratory, where significant genetic changes can be introduced in microorganisms in periods as short as weeks if not days or hours. "Okay", our critics say, "but that is because these organisms are simple and reproduce very quickly." And we reply, "EXACTLY!"

Most problems which we might wish to solve with a GA can be encoded in bit-strings ranging from tens to possibly thousands of bits - certainly simple compared to the information content of a human's set of 23 chromosome pairs each holding approximately 1 x 10^8 base pairs [2]. Furthermore, our computers are fast (and getting faster each year). These artificial bit-string organisms can mate at rates nearing a million per second on the fastest computers. So in the span of, say, 10 minutes, 600 million individuals could have come and gone. Now it is true that in a 100 bit string, the space we are searching greatly exceeds 600 million (2^100 aprox = 10^30 = a trillion trillion). To **exhaustively** search this space, even at a billion searches per second, would take 10^18 seconds or about 30 billion years! "Ha!", our detractors say, "see I told you so!". "Well", we reply coyly, "Of course. **No** algorithm on any computer imaginable could **exhaustively** search such a large space." The power of evolution and genetic algorithms is that they don't search even a small fraction of the possibilities. By using selection of the fittest, crossover, and a random mutation (now and then), they quickly hone in on extremely fit individuals (not always the most fit, but usually in the top 10% or better -- which is more than sufficient for problems of such magnitude).

The idea that I want you to take away from this section is that GA's operate on simply structures very quickly. But because of the *implicit parallelism* (to be defined precisely later in this paper) of evolutionary processes, this quickness is not the primary reason why they do so well. It is the power of natural selection (or more accurately for GA's - artificial selection) that allow them to succeed where other methods fail and to have broader applicability where other methods are narrower in scope.

Guess #3: The Theological Problem

The theological problem is not so much a problem for GA's as it is for biological evolution. However, because biological evolution is a *bad theory* in some quarters it is likely that, to some people, computer based evolution will be guilty by association. The problem stems from the fact that evolutionary theory flies in the face of my peoples religious beliefs. In some, these beliefs are so strong that they will reject any evolutionary explanation no matter how rationally it is presented. Others will see the effectiveness of an evolutionary argument but its conflict with their beliefs makes them uncomfortable.

I will not dwell on this problem further or impose my opinion on the reader. I will, however, leave the reader (who may share these theological objections to evolution) with a final thought. In artificial evolution (i.e., genetic algorithms) there certainly is a *supreme being*. That supreme being is the programmer who writes the GA, loads into the computer and says "let there be light" (or probably something more humble like, "run").

Guess #4: The Hype Problem

As with any new technology, there is a certain amount of hype that surrounds the reality. In some cases, hype can be good because it generates excitement and excitement generates money and money attracts more people and causes growth, and so on. Hype becomes a problem when the technology is not quite ready to live up to the hypster's claims. C++, the object oriented child of the C programming language, is a case where hype proved to be beneficial to its growth. Artificial Intelligence however has been, I believe, hurt by much of the exaggerated claims that flew around in the 60's and again in the 80's with the popularity of expert systems.

Genetic Algorithms have not generated the critical mass that attracts the hypsters in full force. Therefore, this guess is not currently the best one for our question. However, I believe that there is movement in this direction. If the hype generates more money for GA research and applications then that is good. It is our responsibility, as GA practitioners, to either live up to the hype or help keep the hype in check so it does not hurt the long term viability of this field.

Guess #5: The Gambling Problem

When we explain the theory and implementation of a GA it is impossible to proceed very far with out mentioning the word *random*. Now anyone with even limited programming experience knows that computers can generate random numbers (or more accurately pseudo-random numbers). There is no problem there. They even appreciate the fact that random number generators are very useful for such tasks as sampling, simulation and video games. But, if you convey the idea that a search or optimization algorithm is using random techniques, their comfort level drops considerably.

Yes, it is true. GA algorithms employ random numbers in a big way. However, GA's are anything but random search procedures. A GA is an algorithm with a very high probability of producing better and better solutions with each new generation of solutions it creates. A pure random search, in contrast, has a very low probability of finding a good answer on each successive try. And even if it does stumble upon one, there is nothing in this chance encounter which will help it find even better ones. On the other hand, when a GA generates only a few slightly better solutions, it is in a much improved position for generating even better ones on the next try. This successive improvement continues until it hits some local optimal solution or it branches off in search of other solutions as a result of mutation or some other technique. We are not quite ready to prove this fact to the reader, but be assured that the effect can be readily demonstrated with just a small amount of programming.

Yet even when theory, proofs and demonstrations convince the skeptic that GA techniques are certainly viable, they may not be convinced that they are worth using in their problems. Its too much like gambling, even if the odds favor the gambler for a change. The problem with GA's is that they must compete with other techniques that have been honed by years of effort to solve a specific type of hard problem fairly well. If there is a good non-random way to accomplish something it will almost always be preferred over one that is in anyway random. Computer scientists and IS managers usually are not big on gambling, especially when the stakes are high (their company's success, their reputation and their jobs are on the line).

It is my belief that GA's will shine when they have been successfully applied to problems in which all other known techniques have failed or have yielded disappointing results. These problems must be of the kind that require an adaptive algorithm as opposed to a fixed one. They must also be of a kind that exploits the best feature of genetic algorithms, their robustness. Here, the term "robustness" is used to refer to a GA's ability to perform consistently well on a broad range of problem types. Robustness and adaptability are traits that few other algorithmic techniques can claim (neural networks may be one exception). What kinds of problems ultimately require this power? Several may come to mind, but the one I find most exciting is Artificial Intelligence. Intelligence, learning and creativity are traits AI researchers have been trying to impart to computers since the 1950's. Although, in narrow domains, there has been moderate success, a robust reasoning computer is still an object of science fiction. I think it is time for GA's to take a shot at this tough problem.

I will have more to say about GA's and AI in section VII. For now, I will simply state that GA's have an acceptance problem partially because they are still orphans competing for a home of their own. In the areas which they have competed to date, the competition is rough. However, by competing in well understood domains (like function optimization and combinatorial search) more is learned about them and better techniques are discovered. But until they find a home of their own, they will only remain, at most, objects of curiosity to most of the corporate and computer science world.

In attempting to explain (in my own humble viewpoint) why GA's are not yet a household word, I hope you have at least gained an intuitive sense of what they are and their potential power. Section III, will greatly expand on this foundation. The important points from this section are:

- Genetic algorithms are procedures for optimization, search and adaptation.
- They mimic natural selection and biological evolution to achieve their power.
- They are fairly efficient compared to many other methods and are much more efficient than exhaustive search.
- They have no stance or opinion on matters of religion or theology and only ask to be accepted at their face value.
- Genetic algorithms deserve your attention, but please try not to get too excited too quickly lest people think the field is full of fanatics and hypsters.
- They confess to employ random techniques but vehemently deny being gamblers. They work by exploiting information glommed from past experience and not by a simple toss of the dice.

III. Please Understand Me - How and Why GA's Work.

In this section I will dig a little deeper into the mechanics of genetic algorithms. You will see that there are many variations to the "standard" genetic algorithm but that these are all united by the common thread that was contained in our initial simple definition. Specifically, the selection of individuals with simple underlying structure (strings) from a population with a preference for the fitter. Followed by a recombination of these structures to yield a new generation of individuals with higher average fitness.

To facilitate the discussion in this section we will introduce some terminology that you should become familiar with if you wish to do any work with GA's. These terms are gene, allele and locus. A gene (also referred to as a feature, character or detector) refers to a specific attribute that is encoded in the genotype's chromosome (string). The particular values the gene can take are called its alleles. The position of the gene in the chromosome is its locus. A common example taken from biology is that eye color is determined by a gene, the different eye colors are the gene's alleles (blue, brown, etc.) and where the eye color gene happens to lie in the chromosome is its locus. So why don't we just say attribute, value and position? The reason is that GA's are so heavily grounded in natural genetics (with many researchers coming from this domain rather than from computer science) that the terminology of biology seems to have won out over that of computer science (at least as measured by most of the GA papers and books I have read). A lingo can sometimes hinder the growth of a field of study, however it at least helps to identify the research papers of the field as belonging to it and it gives the field some personality.

We will now discuss the characteristic an algorithm must have in order to be considered genetic. We will describe those characteristics and briefly explain how they are usually implemented as well as hint at why they work. A more in depth discussion of why they work is contained in section V.

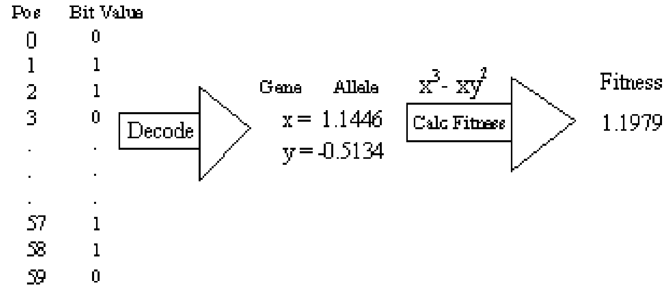
A. Fitness

Because of the nature of a genetic algorithm it is easier to talk about them solving optimization problems which search for maximums. However, this does not prevent us from using GA’s to find minimal values. A minimization problem can always be mapped mathematically into a maximization problem. To understand this we will first talk about a crucial component of all GA’s - the fitness function (also called the objective function).

The fitness or objective function is used to map the individual’s bit strings into a positive number which is called the individual’s fitness. There are two steps involved in this mapping (however in some problems these two steps are essentially accomplished as one). The first step we will call "decoding" and the second, "calculating fitness".

To understand decoding it helps to partition individuals into two parts commonly called the *genotype* or *genome* and the *phenotype*. These terms are borrowed from biology. The genotype, as its name implies, specifically refers to an individual’s genetic structure or for our purpose, the individual’s bit string(s). The phenotype refers to the observable appearance of an individual (pheno comes from Greek for "to show" - *phainein*). For our purpose, the phenotype is the desired result -- the parameters of the problem that yield its fitness. Therefore, decoding refers to the process of mapping the genotype to the phenotype -- the code to the parameters. For example, in the function optimization problem f(x,y) = x^3 - xy^2, we might use a 60 bit string to encode the parameters x and y (30 bits per parameter). The decoding process would then refer to the mapping of these bit strings to real values (x,y) on some range of interest [M,N] (for example [-1.0, 2.0]). Using the genetic lingo x and y are genes which occur at locus 0 and 30 in the chromosome. The gene’s alleles range from -1 to 2.0 in increments of (2 - (-1)) x 2^-30 = 3 x 2^-30.

Example of Fitness Calculation for Parameter Optimization (f(x,y) = x^3 - xy^2



The second step, fitness calculation, is simple once the genotype has been decoded. We simply use a function to map the phenotype’s parameter values into a positive number, the fitness. In a function maximization problem, the fitness function is often the same as the function we are optimizing (provided the domain (output) of the function is positive). If the domain contains negative values or if we are considering a minimization problem then we simply map the output of the function into the required form. For negative values, this will simply involve adding a constant to the output so it is shifted into the positive range. For minimization problems, we make use of the following fact:

Min (f(x)) = Max(g(x)) = Max(- f(x)).

This states that the minimum of a function, f(x), is equivalent to the maximum of some other function g(x) where one specific g(x) for which that is true is -1 x f(x). Again if, -f(x) is negative (it might not be if the domain of f(x) is negative) we can simply add an appropriate constant, C.

Min (f(x)) = Max(g(x) + C) = Max(- f(x) + C).

The operations of decoding and fitness calculation apply even if we are not dealing with a function optimization problem. For example, in the Traveling Salesman* Problem3, the genotype would be the way we choose to encode a tour in a string. The "parameters" or phenotype would be the actual tour being considered and the fitness would be a function of the length of the tour. We need a function of the length of the tour, and not simply the length, because the traveling salesman is a minimization problem.

The reader may be wondering why we always insist on solving a maximization problem and why we require positive values for fitness. To understand this, we must explore another crucial component of genetic algorithms, selection.

B. Selection

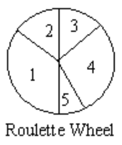
Selection is one of the most important elements of all GA’s. Selection determines which individuals in the population will have all or some of its "genetic material" passed on to the next generation of individuals. The object of the selection method employed in a GA is to give exponentially increasing trials to the fittest individuals. The most common way in which this is accomplished is by a technique called "roulette-wheel" selection. As you will see, it is the implementation of roulette-wheel selection which necessitates positive fitness values where higher values indicate greater fitness. Roulette wheel selection gets its name from the fact that the algorithm works like a roulette wheel in which each slot on the wheel is paired with an individual in the population. This is done such that the size of each slot is proportional to the corresponding individuals fitness. It should be obvious then that maximization problems fit directly into this paradigm - larger slot implies larger fitness. Negative values are not allowed because how can you have a slot of negative size?

A common way to implement roulette wheel selection is to:

1. Sum up all the fitness values in the current population, call this value SumFitness. SumFitness is in effect the total area of the roulette wheel.
2. Generate a random number between 0 and 1, called Rand.
3. Multiply SumFitness by Rand to get a number between 0 and SumFitness which we will call RouletteValue (RouletteValue = SumFitnesss x Rand). Think of this value as the distance the imaginary roulette ball travels before falling into a slot.
4. Finally we sum up the fitness values (slot sizes) of the individuals in the population until we reach an individual which makes this partial sum greater or equal to RouletteValue. This will then be the individual that is selected.

It is not always intuitively obvious that this algorithm actually implements a weighted roulette wheel. To see that it does lets look at some extreme situations. Imagine an individual, I, whose fitness is equal to SumFitness (implying all other individuals have a fitness of zero).

Clearly no matter what number is generated for RouletteValue, *I* will always throw the partial sum over the top, thus having a selection probability of 1. This corresponds to a roulette wheel with just one slot. On the other extreme, an individual, *i*, with fitness zero can never cause the partial sum to become greater than RouletteValue, so it has a zero probability of getting selected. This corresponds to a slot that does not exist on the wheel. All other individuals between these extremes will have a probability of throwing the partial sum over the top that is proportional to their size, which is exactly how we would expect a weighted roulette wheel to behave.



Individual	Fitness	Slot Size %
1	30	35
2	10	12
3	15	18
4	25	29
5	5	6
SumFitness:	85	

Roulette Wheel Selection with Five Individuals of Varying Fitness.

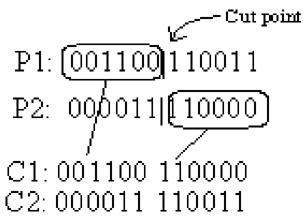
Roulette wheel selection is not the best way to implement selection from the stand point of both efficiency and fairness. The method is not always fair, because random fluctuations (referred to as stochastic errors) can cause the actual number of individuals of varying fitness to receive more or less than their expected allotment. To Combat this, other selection methods have been devised. We will not discuss these here, however. The important quality of all legitimate GA selection techniques is to reward fitter individuals by letting them reproduce more often. This is one of the important ways in which a GA differs from random search.

C. Reproduction

The second critical attribute of all genetic algorithms is that they contain some sort of reproduction procedure. It is here that two individuals selected in the previous step are allowed to mate to produce offspring. This mating is done by a genetic operator commonly called crossover. Crossover is the process by which the bit-strings of two parent individuals combine to produce two child individuals.

There are many ways in which crossover can be implemented. Some of the ways are broadly applicable to all types of problems and others are highly problem specific. Here we will talk about the most primitive (but also highly effective) form of crossover, *single-point crossover*. Single point crossover starts by selecting a random position on the bit string, called a cut point or cross point. The bits from the left of the cut point on parent1 are combined with the bits from the right of the cut point in parent 2 to form child1. The opposite segments are combined to form child2.

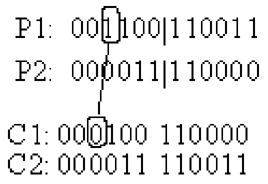
Example of crossover:



Thus child1 and child2 will tend to be different from either of their parents yet retain some features of both. If the parents each had high fitness (which is likely by the fact that they were selected) then there is a good chance that at least one of the children is as fit or better than either parent. If this is the case, then selection will favor this child’s s procreation, if not than selection will favor the child’s extinction.

There is of course a possibility (albeit small) that most or all of the crosses produce children of less fitness. To counter this possibility, a parameter, px - the probability of crossover, is introduced. Before crossover is performed a simulated coin is flipped that is biased to come up heads (TRUE) with probability px. If it does, then crossover is performed, if not than the parents are passed into the next generation unchanged. Since without crossover there is no hope for advancement, px is usually high ($0.5 < px < 1.0$).

Another important GA operator is *mutation*. Although mutation is important, it is secondary to crossover. Many people have the erroneous belief that mutation plays the central role in natural evolution. This is simply not the case. The reason is that mutation is more likely to produce harmful or even destructive changes than beneficial ones. An environment with high mutation levels would quickly kill off most if not all of the organisms. In genetic algorithms, high mutation rates cause the algorithm to degenerate to random search.



Example of mutation.

Unlike crossover, mutation is a unary operator - it only acts on one individual at a time. As bits are being copied from a parent individual to a child. a weighted coin is flipped, if it comes up TRUE than the bit is inverted before copying. The probability of the simulated coin coming up TRUE is called pm - the probability of mutation. As previously stated pm is small (0 <= pm <= 0.1).

Another genetic operator is called inversion. Inversion is not used as often as crossover and mutation in most GA's. Inversion is a process that shifts the locus of one or more gene in a chromosome from one point to another. This does not change the meaning of the genotype in the sense that a genotype before and after inversion will still decode to they same phenotype. If this is true, then why bother with inversion at all? The theory behind inversion is that there are groups of two or more genes in a chromosome that work together to yield a high fitness. If these genes are physically close together than single point crossover is much less likely to disturb these groups. Although this argument seems reasonable, inversion used in practice as achieved very mixed results. This is why many GA's ignore inversion all together.

D. Other Kinds of Genetic Algorithms

So far in our discussion we have only considered bit string encodings in our genotypes. However, there is an increasingly large number of GA's being researched and fielded which do not use bit encoding at all. Some researchers who are purists would claim that these are not true genetic algorithms. The term *evolutionary program* is sometimes applied to algorithms which are in the spirit of GA's but use other kinds of data structures. For purposes of our discussion we will lump all of these variations under the category of genetic algorithm.

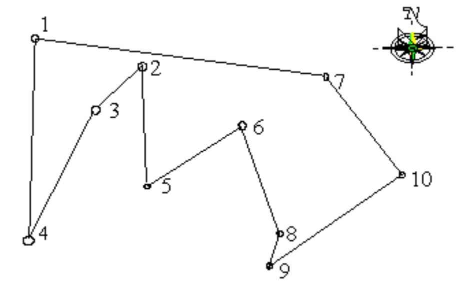
Two common forms of alternate genetic encodings used are real numbered and ordinal. Real encodings use strings (arrays) of real numbers rather than bits. Thus, instead of encoding each parameter in a genetic algorithm as many bits, a single real number can be used. With a real numbered scheme a decoding step is not necessary since the "genes" are already in the form they will be used.

When using a real valued encoding with several genes, traditional single point crossover can certainly be used. However, one of the reasons for going to real encodings is to exploit the ability to manipulate the chromosomes directly as strings of numbers. This suggests that new forms of crossover might prove more useful. One possibility is crossing by averaging the two parents. Another might use weighted averages, weighting each child toward a different parent. Still others may combine features from single point crossover with weighted crossover. Imagination and experimentation can often lead to varying results based on the nature of the problem.

Mutation in real encodings also presents the designer with a variety of choices. The simplest might be to completely replace a real gene with another randomly generated one. One can also average a random number into the existing gene. Another good possibility is to add or subtract a small noise factor into the gene's value. Again the choice and results usually depend on the problem at hand.

An example of a domain where real encodings are useful is in engineering design optimization. Here, there are usually a large number of real parameters which describe the various design choices that can be made. Finding a near optimal design for a given application is often an arduous task. Because of the larger parameter space, bit strings would probably yield poor results. In addition, real encodings would be more intuitive to the engineer and allow him to incorporate other known algorithms into the optimization process. This is known as hybridizing the genetic algorithm.

Another class of optimization problem that begs for a different encoding scheme is combinatorial optimization. In combinatorial optimization, we are concerned with optimizing the order of some operation rather than a set of control parameters. An example of this is the Traveling Salesman Problem (TSM) [3] previously mentioned. In TSM we have a salesperson who must make a tour throughout the country visiting various cities and then returning to home base. The salesperson can not go to any city more than once. The goal is to devise a travel plan (a tour) which minimizes the total distance traveled. It is not obvious that this algorithm can be mapped onto a bit-string encoding, although it has been done. I more natural encoding is to use a string of ordinals. These are simply positive integers which describe the order in which the cites should be visited (assuming each city is associated with a number). There are actually several methods of encoding the TSM. The one illustrated here is properly known as a *path encoding*.



Traveling Salesman Example with an Ordinal Encoded Individual.

Ordinal Genotype: [1,4,3,2,5,6,8,9,10,7] represents a tour.

When working with ordinal encodings, it is necessary to create both crossover and mutation operators that are specific to this form of encoding. The kinds of operators we have considered for bit strings and reals are problematic when applied to ordinals. The reason is that operators like single point crossover or standard mutation are likely to cause the generation of invalid individuals. An ordinal chromosome must not contain duplicate ordinals and must have every ordinal in its range represented.

Many different ordinal operators have appeared in the GA literature. We will not go into this at length here. If you have received this white paper as part of the EOS application framework then you can consult the user manual for more details on these operators. Otherwise, you can consult one of the references in the bibliography.

IV. Yes, I have a Past - A little history for the historically bent.

In this section we give a brief chronology of the figures and events leading up to the present day work in Genetic Algorithms.

- Georges Buffon - mid 1700's Had ideas that some historians believe were precursors to Charles Darwin's concept of evolution. For example, he believed that certain species shared a common ancestor. However, the totality of his published works show that he had much different preconceptions than did Darwin.
- Erasmus Darwin (grandfather of Charles) - 1731-1802 Published a book called Zoonomia (1794-96) which proposed a theory of evolution. However, his views were much different than the younger Darwin in that he assumed that living things were designed by God to be *self improving* over time. These improvements would then be passed on to later generations. There was, however, no mention of natural selection in his work.
- Jean de Lamarck later 1700's to early 1800'sCredited with the *theory of acquired characteristics* which became popular again in the late 19th century. This theory (now believed by most to be wrong) was put forth during Darwin's time as an alternative to natural selection. It claimed that individuals passed on characteristics that they acquired during their lifetime to their offspring.
- Charles Darwin -(1809 - 1882)Generally credited by most historians to be the true discoverer of evolution by natural selection. His basis for discovering this theory arose from the extensive natural history notes he took during the now famous *Voyage of the Beagal* - a 5 year voyage throughout South America, the surrounding islands, Australia and the southern tip of Africa. It was Darwin's insight that small random changes that occurred in living things over time could give them a slight edge over their predators and kin. This edge would cause them to live longer and therefore produce more offspring. Over time the traits that provided a selective advantage would be magnified and allow new species to ultimately arise.
- Gregor Mendel - (1822-1884)Although Darwin discovered natural selection he did not understand the materialistic mechanism which allowed it to work. The origin of this discovery is generally credited to Mendel who discovered the theory of dominant and recessive traits while working with pea plants.
- Discovery of DNA (1869)The origin of genetics is usually associated with the discovery of DNA by Friedrich Miescher.
- Discovery of the structure of DNA (1953)James Watson and Francis Crick are normally credited with the discovery of the double helical structure of DNA. However, many other researcher's work was necessary before hand to allow this discovery to be made. The significance of this discovery was that it allowed the inner workings of the genetic code to begin to be understood. This understanding was crucial to putting evolutionary theory on a more solid ground. Specifically it helped explain how various traits of parent individuals can be passed onto their children and how new traits not found in either parent could arise.
- Neo-Darwinism (1930's - Present)Neo-Darwinism is the synthesis of Darwinian evolution and modern understandings of genetic structure. "At the base of this synthesis is the concept that mutation occurs randomly and furnishes the fuel for evolution by introducing genetic variability. Evolution can then be defined as a change in the frequencies of genes introduced by random mutation with natural selection usually considered the most important cause of such changes."[4]
- John Holland begins his investigation into adaptation in computer programs (1960's)Holland is generally considered the "father of genetic algorithms" although he did not name the child (this was done by Bagley). A series of papers published in 1962 on adaptive systems theory laid the conceptual foundation for the field's development.

- J.D. Bagley coins the term "Genetic Algorithm" (1967)The first published application of genetic algorithms appeared in Bagley's doctoral dissertation. Bagley addressed the application of genetic algorithms to game playing because of the popularity of games in AI at the time.
- R.S. Rosenberg (1967)In a more biologically oriented domain, Rosenberg publishes work on the simulation of single celled organism populations on a computer.
- Adaptation in Natural and Artificial Systems (1975)Holland publishes the first edition of the classic book on the ideas and theory surrounding genetic algorithms. This book discusses his schema theory, the optimal allocation of trials, reproductive plans, genetic operators, the robustness of GA's and other topics related to theory and application.
- K.A. De Jong (1975)De Jong helps place GA's on a more solid experimental foundation by publishing work on a 5 function test bed for genetic algorithms. This test bed contained functions that were considered in some sense to be *GA hard* - difficult for a genetic algorithm to solve. These functions have been used over and over again to benchmark new approaches to GA implementation.
- J.J. Grefenstette writes GENESIS (1984)GENESIS is a system written in the C programming language which has helped foster GA growth and discovery by being made available in the public domain.
- The First International Conference on Genetic Algorithms and their applications (1985)The field gets its own international conference indicating that published and applications research in this field has reached a critical mass necessary for further growth.
- Several text and reference books are published on Genetic Algorithms (1989 - 1992)Another sign that the field is continuing to grow in both acceptance and involvement.
- John Koza 1991Koza receives a U.S. patent for genetic algorithm software related to the production of software via evolutionary means.
- GA's become shrink wrapped (1990 - 1995)Several companies release commercial software packages that allow developers to incorporate the power of genetic algorithms into their applications. One such package, called EOS: Evolutionary Object System (designed by yours truly) is based on the popular object oriented language C++. It exploits the extensibility of object oriented software by providing a rich set of components that can be assembled into a fully functioning GA application while providing a vehicle for the extension of the framework by the user.

There, of course, have been many other people and events which have been of major importance in the evolution of genetic algorithms. A more complete historical perspective can be found in Goldberg's text listed in the bibliography. My apologies for any individuals unfairly left out due to my own biases and constraints on time and space.

V. Nothing Up My Sleeve - A little theory for the theoretically bent.

In this section we will give a cursory analysis of the theory underlying genetic algorithms. I do not claim to be an authority on GA theory so readers should use this section as a gentle introduction and refer to the books I have listed in the bibliography for more complete coverage.

A. The Fundamental Theorem of Genetic Algorithms.

Why do genetic algorithms work so well? Even if you have read the previous sections carefully and understand the mechanics of a GA, you may still feel uneasy because you are not quite sure why they work. We will attempt to remedy this.

In section, III ,we stated that GA's work because a GA gives exponentially increasing trials to the fittest individuals. This is not completely accurate. For if a GA only worked at the level of the individual, how could it hope to explore spaces as large as 2100 (1030) or greater. Certainly to be efficient, a GA must limit its population size to some manageable number. This number is usually in the range of 102 to 105. How could a few hundred or even a few thousand individuals explore a space with a trillion trillion different points? The answer lies in a more accurate definition of the underlying GA theory. Instead of saying that individuals receive exponentially increasing trials, ***the fundamental theorem of genetic algorithms*** states that each time an individual is processed the GA is not simply sampling 1 point in the space but many many points. To understand how this can be we must understand the concept of *schemata*. This will be explained next.

B. Schema Theorem and The Building Block Hypothesis

When considering bit string genotypes, we are of course limiting each position in the string to the characters 0 and 1. For the purpose of this discussion we will add a third symbol to this alphabet and call it the *don't care* symbol (*). This gives us the new alphabet {0,1,*}. A string from this alphabet might look like the following: 00**11. The meaning of the don't care symbol, as its name implies, is that we don't care what value (0 or 1) is contained at that position. Another way of stating this is that the sample string, 00**11 *matches* all of the strings in the following set {000011, 000111, 001011, 001111}. Strings which include the don't care symbol are called *schemata* or *similarity templates*. To see why schemata are important, let's look at a hypothetical population of bit-strings and their associated fitness.

String	Fitness
10011	361
00110	36
11000	576
01110	196

Without even knowing how the strings decode to yield their fitness values you can immediately see some patterns between the strings and the their fitness. One pattern that seems to emerge is that strings which begin with 1 have significantly higher fitness values than those which begin with zero. Another observation is that strings with more 1's than zero's tend to have higher fitness although our first observation seems to be the more significant of the two. So by only looking at 5 strings out of a possible 32 we have been able to extract a lot of information. If one had to make a guess as to a string that would give even better fitness, who would not guess 11111, given the above two observations? Our first observation is equivalent to stating that strings which match the schema 1**** have higher fitness then those that match the schema 0****. The second observation is that strings which match schemata from the set {1111*, 111*1, 11*11, 1*111, *1111, 111**, 11**1, 1**11,**111} will have greater **average** fitness than those which match schemata in the set {0000*, 000*0, 00*00, 0*000, *0000, 000**, 00**0, 0**00,**000}.

The basis, then, for the success of GA's is that they implicitly search for patterns (schemata) with higher fitness. In a bit string of length 5 there are only 2^5 = 32 individuals but 3^5 = 243 schemata. Every time a GA processes a single individual it is actually processing many schemata. How many? Well according to the schema theorem first worked out by John Holland, in a population of size N the number of schemata effectively processed is on the order of N^3. The mathematical derivation of this result is beyond the scope of this paper. Interested readers should turn to the bibliography for guidance on where they can see this result proven.

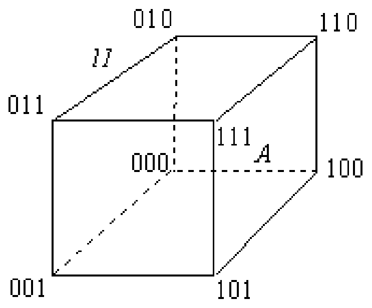
It is this power to process large numbers of schemata with each generation that give GA's the *implicit parallelism* that we mentioned in section III. We can, therefore, see how a population of, say, 1000 individuals of length 100 is in a good position to explore a space of 10^30 points. The number of schemata processed in each generation is approximately 1000^3 = 10^9. Add to this the fact that bad schemata are quickly weeded out of the search by selection, and we have a fairly powerful search procedure indeed.

A related observation that emerges when one considers the relationship between schemata and the crossover operator is called the *building block hypothesis*. It states that genetic algorithms seek near-optimal performance through the juxtaposition of short, low-order, high-performance schemata called *building blocks*. In this definition short refers to the length of the defining portion of a schema. This is the distance between the first 0 or 1 and the last 0 or 1. For example, in 1***0*, the length is 4 and in **111* it is only 2. Order (as in low-order) refers to the number of fixed positions (0 or 1) in the schema. Schema of low order match a larger number of individuals than those of high order. A *high performance schema* is one that matches individuals of high fitness.

Considering that crossover tends to chop up schemata as it recombines individuals, it is clear that short schemata have a higher chance of surviving crossover intact. Low order schema effectively process more individuals and are also more likely to remain intact. The building block hypothesis has application in both the design of new kinds of crossover operators and in determining how to best encode a problem.

C. Hyper-planes

Another view of genetic algorithms that may be more intuitive to those who prefer to think spatially, is the notion of *hyper-planes*. A hyper-plane is a multidimensional analog to a Euclidean 2 dimensional plane. This is most easily visualized by considering a toy problem with a bit string of length 3. This



space can easily be represented geometrically.

In this 3D space the schema 01* represents the line labeled *l1* in the figure and the schema 1** represents the right side of the box (labeled *A*). Of course, the schema *** represents the entire box. In higher dimensional spaces (i.e. spaces with strings of greater length) schema would correspond to what mathematicians call hyper-planes. Using this paradigm, a GA can be considered to be "cutting across different hyper-planes" in its search for more optimal solutions. This is in contrast to a random or exhaustive procedure which can only proceed from point to point.

VI. Applications of Genetic Algorithms

In this section we list some GA applications that we know of that range across a broad cross section of domains. We briefly describe the domain and the application. There are of course many more applications that could be added to this list and the number is growing every day. We hope that in these 10 domains you will find at least one that is of some interest to your possible GA needs. Other applications can be found in the books listed in the bibliography section.

1. Scheduling
Scheduling can be defined as a problem of finding the optimal sequence to execute a finite set of operations such that a certain set of constraints are not violated. A scheduler usually attempts to maximize the utilization of individuals or machinery and minimize the time required to complete the entire process being scheduled. Conflicts can arise when an individual or machine is scheduled for more than 1 operation at a given time or when the schedule utilizes more resources than are available. Other constrains include priority of some tasks over others and precedence of certain tasks with respect to others.

Descriptions of GA applications in the domain of scheduling can be found in the following papers.

- Schedule Optimization Using Genetic Algorithms by Gilbert Syswerda in Handbook of Genetic Algorithms edited by Lawrence Davis. Describes the scheduling activities in a laboratory and how a GA can be applied. His GA uses order based encodings and is a good example for readers interested in these encodings.
- Job Shop Scheduling with Genetic Algorithms by Lawrence Davis in Proceedings of the First International Conference on Genetic Algorithms and their Applications edited by John J. Grefenstette. Describes the scheduling of work in a job shop composed of a number of workstations. A simple GA and encoding scheme is outlined that optimizes the scheduling of jobs at the stations to minimize cost and maximize profit.
- A Classifier[5]-Based System for Discovering Scheduling Heuristics by M.R. Hilliard and G.E. Liepins in Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms edited by John J. Grefenstette. This paper shows how classifier systems can indirectly improve job shop scheduling by discovering heuristics (rules of thumb) that can be applied to the problem domain.
- Using Genetic Algorithms to Schedule Flow Shop Releases by Gary A. Cleveland and Stephen F. Smith in Proceedings of the Third International Conference on Genetic Algorithms edited by J. David Schaffer. This paper contains a comparative analysis of various GA techniques that have been proposed for industrial scheduling problems.
- A System for Learning Routes and Schedules with Genetic Algorithms by P.S. Gabbert, et. al. in Proceedings of the Fourth International Conference on Genetic Algorithms edited by Richard K. Belew and Lashon B. Booker. Describes a GA approach to routing and scheduling freight trains.
- Conventional Genetic Algorithm for Job Shop Problems by Ryohei Nakano in Proceedings of the Fourth International Conference on Genetic Algorithms edited by Richard K. Belew and Lashon B. Booker. Explains how a conventional bit-string genetic algorithm can be applied in the job shop domain.
- The Application of Genetic Algorithms to Resource Scheduling by Gilbert Syswerda and Jeff Palmucci in Proceedings of the Fourth International Conference on Genetic Algorithms edited by Richard K. Belew and Lashon B. Booker. Describes a hybrid genetic system (i.e., one that uses a GA and a conventional algorithm) for resource scheduling in a laboratory.

2. Economics and Finance
Finance is a broad domain. Some of the possible problems in this domain that are applicable to genetic

algorithms are: Economic Forecasting, Credit Approval, and Investment Analysis.

- In Holland’s Adaptation in Natural and Artificial Systems he discusses applications of adaptive systems toward economic forecasting.
- A Genetic System for Learning Models of Consumer Choice. by David P. Greene and Stephen Smith in Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms edited by John J. Grefenstette. This work explains how a GA can be used to discover rules that people use to identify acceptable and unacceptable products.
- Man Machine Interfaces, Inc. (unpublished work) has constructed several prototype GA systems for discovering rules which can be applied to making buy and sell decisions in stock market trading. The system was designed to constantly adapt to changing market conditions.
- An article called, The New Rocket Science appeared in Business Week (11/2/92) that explains how GA’s, Neural Nets and Chaos Theory are being applied on Wall Street.

3. Software Engineering
Possible GA applications in this domain include software synthesis and software optimization.

- Using the Genetic Algorithm to Generate LISP Source Code to Solve the Prisoner’s Dilemma [6]. by Cory Fujiko and John Dickinson in Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms edited by John J. Grefenstette. A description of a GA that is able to manipulate LISP code toward the production of solutions to the Prisoners Dilemma.
- Evolving a Computer Program to Generate Random Numbers Using the Genetic Programming Paradigm by John R. Koza in Proceedings of the Fourth International Conference on Genetic Algorithms edited by Richard K. Belew and Lashon B. Booker. Demonstrates that random number generators can be genetically breed with near optimal performance.
- A Genetic Algorithm for Database Query Optimization by K Bennett, M. Ferris and Y. Ioannidis in Proceedings of the Fourth International Conference on Genetic Algorithms edited by Richard K. Belew and Lashon B. Booker. Presents a method for applying adaptive GA’s to performing complex query optimization tasks in a relational database.

4. Graphics
Often in a graphics systems it is desirable to automatically arrange graphic objects on the screen according to some aesthetic criteria. This is especially applicable to Computer Aided Software Engineering (CASE) and Computer Aided Design (CAD) applications. GA’s can be used to optimize graphic layout in these applications.

- How to Draw a Directed Graph by P. Eades and X. Lin described in Genetic Algorithms + Data Structures = Evolution Programs by Zbigniew Michalewicz. Describes how a GA can be used to produce aesthetically pleasing drawing of directed graphs on a computer screen. This work has obvious application to the production of CASE tools.
- Graphic layout of iconic elements in a objected oriented programming tool by S. Mangano. Unpublished, contracted research by Man Machine Interfaces into the application of GA’s to visual programming environments.

5. NP Complete Problems
Various forms of complex problems (referred to as Non-Deterministic Polynomial Complete) have been addressed using GA’s. These include the traveling salesman, map coloring and truth assignment

problems.

- The Traveling Salesman and Sequence Scheduling: Quality Solutions Using Genetic Edge Recombination by D Whitley, T. Starkweather and D. Shaner in Handbook of Genetic Algorithms edited by Lawrence Davis.
- Genetic Algorithms for the Traveling Salesman Problem by J.J. Grefenstette, R. Gopal, B. Rosmaita and D. Van Gught in Proceedings of the First International Conference on Genetic Algorithms and their Applications edited by John J. Grefenstette.
- Using Genetic Algorithms to Solve NP Complete Problems by Kenneth De Jong and William M. Spears in Proceedings of the Third International Conference on Genetic Algorithms edited by J. David Schaffer.
- Genetic Algorithms + Data Structures = Evolution Programs by Zbigniew Michalewicz. contains several chapters devoted to solving NP complete problems using GA's.

6. Robotics
GA's have been applied to optimizing robot movement.

- A Genetic Algorithm Applied to Robot Trajectory Generation by Yuval Davidor in Handbook of Genetic Algorithms edited by Lawrence Davis. Describes the control of a robot arm using a genetic algorithm to search for optimal trajectories.

7. Telecommunications
Routing of information through a complex communications network is an extremely important problem in today's world full of computer networks, cellular networks, packet switching networks and the like. This is a tough problem that some researchers have addressed using GA's.

- Genetic Algorithms in Communication Link Speed Design by Lawrence Davis and Susan Coombs in Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms edited by John J. Grefenstette. GA applied to the task of finding low cost sets of packet switching communication network links.
- Dynamic Anticipatory Routing in Circuit-Switched Telecommunications Networks by L. Cox, Jr., L. Davis and Y. Qiu in Handbook of Genetic Algorithms edited by Lawrence Davis.

8. Forecasting
A forecasting problem can be described as a problem of predicting future behavior from historical data. We have already mentioned financial forecasting in item 2. However, other types of forecasting problems are applicable to GA's (Weather, earthquakes, horse racing, ...).

- A SEAGUL Visits the Race Track by Michael de la Maza in Proceedings of the Third International Conference on Genetic Algorithms edited by J. David Schaffer. A GA system for handicapping horses based on historical data.

9. Engineering and Design
Optimizing a design that includes many different parameters is a time consuming and difficult task for even expert engineers. GA's can assist in design by quickly narrowing in on close to optimal designs given certain user specified design constraints.

- Genetic Algorithms in Parametric Design of Aircraft by Mark Bramlette and E. Bouchard in

Handbook of Genetic Algorithms edited by Lawrence Davis. Explains how a GA can be used to optimized the parameters governing certain aspects of air craft design.

- Interdigitation: A Hybrid Technique for Engineering Design Optimization Employing Genetic Algorithms, Expert Systems and Numerical Optimization by D. Powell, M. Skolnick and S. Tong in Handbook of Genetic Algorithms edited by Lawrence Davis. Describes a novel technique for combining GA's with other technologies. The technique utilizes expert systems and numerical optimization to narrow the GA's search down to areas that an experienced engineer knows are promising.

10. Neural Networks

Neural networks are computer programs which simulate the workings of the nervous system (in a much simplified form) in order to solve various problems. Many of the problems that neural nets have been applied to are similar to those listed here for GA applications. However, GA's have also been applied to creating better neural networks. Applications include both the training of a neural network and the optimal design of a neural network.

- A conectionist algorithm for genetic search by David Ackley in Proceedings of the First International Conference on Genetic Algorithms and their Applications edited by John J. Grefenstette. A architecture for function maximization is proposed which uses principles from neural nets and GA's.
- Towards the Genetic Synthesis of Neural Networks by S. Harp, T. Samad and A Guha. Designing Neural Networks Using Genetic Algorithms by G. Miller, P. Todd and S. Hegde both in Proceedings of the Third International Conference on Genetic Algorithms edited by J. David Schaffer. Explain how GA's can be used to find optimal neural net architectures.
- Genetic Reinforcement Learning with Multi-layer Neural Networks by D. Whiley, S. Dominic and R. Das in Proceedings of the Fourth International Conference on Genetic Algorithms edited by Richard K. Belew and Lashon B. Booker. Displays how "genetic hill climbers" can be used for learning connection weights in a neural network.

VII. Predictions and Wild Ass Guesses.

In this section I will go out on an evolutionary limb and make some predictions as to where I think GA technology is going and how it will affect the rest of the computer industry. Prediction is always a dangerous thing. For one, there is a good chance the forecaster will be wrong. Secondly, there is a fine line between prediction and hype. In the 60's all kinds of grandiose claims were made for AI, similar claims were made in the 70's and again in the 80's. Only a small fraction of these claims have come to fruition and often with much downplayed abilities than was claimed. One of the mistakes that AI forecasters have made was to put specific dates on when they thought things would happen. "In 10 years computers will be able to do X", in 15 years a program will be better at Y than any human", etc. Many of these predictions will probably come true, but asserting when they will creates false expectations and negative feelings when the software fails to appear.

In this section I will be more humble and state my predictions with out hard time frames. Some might call this chicken, buy the purpose here is not to demonstrate my fortune telling skills, but to stimulate the readers imagination.

Prediction #1 - GA's will not go away.

This is the safest of the seven predictions I make here. I simply claim that GA theory and application is not a passing fad but will continue to grow to the point where GA based systems are as common place as more traditional software.

Prediction #2 - CAGAD Systems will appear.

Computer Aided Genetic Algorithm Development (CAGAD) Systems will appear in which GA based systems can be built with no direct computer programming. An implentation of such a shell has been produced by Man Machine Interfaces. It is called the Genetic Object Designer. It still requires some level of programmer but in a much nicer environment with a iterpreter and code generator. Later version of the Genetic Object Designer will make creating GA applications even easier. The Genetic Object Designer (and similar systems) will bring the technology closer to the people who are most likely to benefit from it - engineers, financial analysts and other technical types without the time or desire to program.

Prediction #3 - The GA field will experience rapid growth with parallel processing systems.

Parallel processing computers have been on the market for some time now. However, they are only just beginning to achieve wide spread acceptance. The most commonly used parallel systems have only 2 to four processors. As systems enter the mainstream with 100 to 1000 processors, the algorithms best in position to exploit this power will be GA's. Because of the fact that a GA works on a population of individuals, it is a parallel processing natural. I believe that not only will GA's benefit from parallel computing but successful GA's will drive the demand for this power.

Prediction #4 - GA's will be an inherent part of successful AI

This is prediction is perhaps the most chancy of the ones considered so far (but wait I'm just getting warmed up!). I believe that much of the principals discovered by AI researchers about logic, inference an other reasoning mechanism will be replaced by a mechanism based on a GA. I even have an outline in mind for how such a system would work. That is a topic for another time, however. For now I will simply give a somewhat sketchy justification for this prediction.

Creativity is, in a strong sense, the Holy Grail of AI. Most would agree that as humans we see creativity to be the pinnacle of intelligent behavior. Computer systems built today simply do not display creative behavior. In certain limited domains, computers have been programmed to display some creativity. However, the wisps of creative behavior displayed by these systems quickly goes stale after an extended exposure to them. Human creativity, on the other hand, seems to be an inexhaustible well, especially in the most gifted individuals.

Yet even though humans are masters of creativity in the animal kingdom, there exists a creative force which makes even the creativity of the human geniuses pale by comparison. This is, of course, the creativity displayed by nature. There has never been a time when I have picked up a National Geographic or tuned into a PBS special where I have not be simply amazed at the creativity of Nature. Who amongst us, could ever claim to be able to have designed even a fraction of the wonder we see in the natural world? Now if we believe (as I and many scientists do) that this splendor is the work of natural selection, how can we deny GA's (being based in the same principles) the latent power to exhibit creativity? I do not think we can. Certainly, the field is currently nowhere near living up to this claim, but I am quite sure that it has the underpinnings necessary to reach this point. At least, I think its prospects are much better than any system based on logic and inference mechanisms. But enough said. Those who think I am "just so full of it" will not be convinced in the span of a few more sentences.

Prediction #5 - GA's will change the face of Software Engineering

Regardless of the success of GA's in AI and creativity in general, I strongly believe genetic algorithms will change the way a large percentage of conventional computer systems are developed. The seeds of this change have already been planted. Research by Koza and others have demonstrated genetic based systems that can piece together complete and functional programs from collections of small subroutines. I believe that systems of this kind will get much better with time. And that the day is coming where as much as 20% of the computer software in use will have been bred rather than written by hand.

Prediction #6 - GA's and Neural Networks will be unified by discoveries in neurobiology.

Perhaps the biggest proponent of this view point (although not exactly in the form here) is Dr. Gerald Edeleman. In his books Neural Darwinism and Bright Air, Brilliant Fire he exposes a theory where intelligence and consciousness is explained by underlying evolutionary processes occurring in the brains circuitry. His theories are not generally accepted (and are even disdained by many other respected scientist) but if correct, they foreshadow a unification of GA's and neural networks in the realm of computers. This unification will reach much deeper than the current applications of GA's to the training and design of artificial neural networks. It will show that neural like circuitry is the ideal medium in which to implement evolutionary algorithms. Although I make this prediction, I am somewhat less sold on it than my previous ones. I am not usually comfortable with expounding theories which I can only imagine in the sketchiest of details how they might work. Still, given that I believe strongly in prediction #4, it seems unlikely that two completely orthogonal solutions to creativity exist, one based in neural path ways and the other in evolutionary processes.

Prediction #7 - Computer architecture will be experience a radical transformation.

This prediction in and of itself has little risk associated with it. Of course computers will change. The prediction I make here goes a little further in that it states how and why. As we have already mentioned, the onslaught of massively parallel computers is just over the horizon. But I believe that the generation of computers to come will not simply be more processors of the same underlying architecture, but more processors of a much radically different kind of architecture supporting a much different kind of computation. This architecture will consists of much simpler building blocks than the microprocessors we use today. They will not be based in electricity, but probably in optics. They will not run strictly mechanical programs based on ridged control structures, but rather they will execute much more fluid computations where the desired behavior is emergent rather than pre-planned. In short, computers will look and behave a lot less like machines and a lot more like organisms. This is not to say that wet ware based on carbon chain chemistry and polymers will rule out over silicon based components. But that the form computers will take will be more architecturally similar to living information processors than to today's microprocessors.

Annotated Bibliography

- Antonoff, Michael. Genetic Algorithms: software by natural selection in Popular Science, Oct. 1991, p.70. Contains introduction to GA's and description of John Koza's work.
- Bauer, Richard J. Genetic Algorithms and Investment Strategies. John Wiely & Sons.

1994.Explains how GAs can be used to devlope finacial trading systems. Includes introduction to GA's, as well.

- Belew, Richard K and Booker, Lashon B., editors. Proceedings of the fourth international conference on genetic algorithms. Morgan Kaufmann Publishers, 1991.These conference proceedings cover the following categories:
    1. Representation and genetic operators
    2. Genetic algorithm techniques and behavior
    3. Formal analysis of genetic algorithms
    4. Parallel genetic algorithms
    5. Classifier systems and other rule based approaches
    6. Genetic algorithms in hybrid methods
    7. Genetic algorithm applications
    8. Connectionism and artificial life
- Bowler, Peter J. Evolution: The history of an idea. University of California Press,1989.An insightful survey of the people and events that lead up to and followed Darwin's discovery. Contains no material related to GA history, just biological evolution.
- Davis, Lawrence, editor. Handbook of genetic algorithms. Van Nostrand Reinhold, 1991.Excellent introduction to genetic algorithms with a bias towards application rather than theory. Explains principals of GA hybridization and alternative encoding schemes. Includes detailed case studies in the application of GA's in a wide cross section of domains.
- Forrest, Stephanie, editor. Emergent Computation. The MIT Press, 1991. A series of technical essays on topics related to genetic algorithms and artificial life. Provides an excellent (though fairly technical) exposure to the wider field of emergent computation of which GA's can be considered a sub-discipline.
- Goldberg, David E. Genetic algorithms in search, optimization and machine learning. Addison-Wesley Publishing Company, Inc., 1989. A practical introduction to genetic algorithms and classifier systems. Contains sample code in Pascal for a fully functional (yet rudimentary) genetic algorithm and a simple classifier system. Some of the textual explanations of the underlying mechanisms are difficult to understand from a implementation perspective, however.
- Grefenstette, John J., Proceedings of the first international conference on genetic algorithms and their applications. Lawrence Erlbaum Associates, Publishers, 1985. These conference proceedings cover the following categories:
    1. Theory of genetic algorithms
    2. Classifier systems
    3. Early Applications
- Grefenstette, John J., Genetic algorithms and their applications: Proceedings of the second international conference on genetic algorithms . Lawrence Erlbaum Associates, Publishers, 1987. These conference proceedings cover the following categories:
    1. Genetic search theory.
    2. Adaptive search operations.
    3. Representation issues (genetic encoding).
    4. Connectionism and parallelism.
    5. Credit assignment and learning.
    6. Applications.
- Holland, John H. Adaptation in natural and artificial systems. An introductory analysis with applications to biology, control and artificial intelligence. Second edition, The MIT Press,1992.The field's classic work. Emphasis of this book is on theory. Assumes a solid background in combinatorial mathematics, probability and some calculus. Text is clear enough

that readers with out the prerequisite mathematical background can still benefit from its reading.

- Michalewicz, Zbigniew. Genetic Algorithms + Data structures = Evolutionary Programs. Springer-Verlag, 1992. Contains some of the clearest explanations of various underlying algorithms used in various GA methodologies. Large emphasis on combinatorial problems and non standard encodings. Contains several application examples.
- Schaffer, J. David, editor. Proceedings of the third international conference on genetic algorithms. Morgan Kaufmann Publishers, 1989.These conference proceedings cover the following categories:
    1. Genetic algorithm theory.
    2. Applications.
    3. Classifier systems.
    4. Neural networks.
    5. Parallel genetic algorithms.
- Singer, Maxine and Berg, Paul. Genes & Genomes: A changing perspective. University Science Books, 1991. Genetic textbook describing in detail the molecular and biological basis for heredity, genetic expression and engineering. Provides insights into the genetic code which possibly may stimulate the readers ideas on artificial genetic encodings.
- Strickberger, Monroe W. Evolution. Jones and Bartlett Publishers, 1990. A good textbook on the theory of biological evolution. Covers history of, physical and chemical basis for, organic basis of and mechanics of evolution. Excellent introduction for individuals looking for inspiration from nature in order to improve GA theory and practice.

Notes

1. *I have, of course made these terms up, to avoid saying selection, crossover and mutation.*
2. *A base pair is a unit more comparable to a bit in information content than a gene. There are two kinds of base pairs AT and CG. 'A' stands for adenine, 'T' for thymine, 'C' for cytosine and 'G' for guanine. A gene is usually in the order of 10^3 base pairs.*
3. *We use the historically correct name for this problem rather than the more equitable "Traveling Salesperson.". For those unfamiliar with the TSM, it is explained in subsection D, of this section.*
4. *Quote from, Evolution. Monroe, W. Strickberger. 1990, Jones and Bartlett Publishers, Inc. pg. 418*
5. *Although I did not discuss classifier systems in this white paper, a significant amount of research goes into these types of systems which use a GA for machine learning. See the bibliography for more information.*
6. *The Prisoner's Dilemma is a problem from game theory. See paper for details.*