A Recessive/Dominant Genetic Algorithm

David Bookstaber

Introduction to Genetic Algorithms

Genetic algorithms (GA's) attempt to apply the theory of evolution to problem solving in an effort to evolve solutions to problems that are either too complicated to deal with directly, or too computationally intensive to fully investigate. The first GA's, developed in the late 1950's, endeavored to do this exclusively through mutation. In the early 1960's, it was tried through a sort of mating of partial solutions to arrive at complete ones. The GA's used today have been attributed to the work of John Holland, who combined mutation and mating in the same program in the mid-1960's.

GA's have found numerous applications in optimization of real-life models, search for solutions to complex problems, and adaptation of programs to new situations. For example, General Electric was designing a new jet engine turbine that, on a computer, had 10^{387} possible designs. Researchers applied a GA to the problem and in less than two days found a design that was 60% more efficient than the best generated by their other programs. David Goldberg, an authority on GA's, applied one to a model of a natural gas pipe network. In reality, this network has been controlled by humans because no program could be written that could take into account all of the factors involved in the system. Goldberg's GA, after a training period, was able to deliver gas in a simulation of the network with efficiency approaching that of humans in practice, and was also able to respond properly to leaks. One popular application of GA's is to the Traveling Salesman problem: Given a number of cities at fixed locations, the GA finds the shortest route possible that includes every city. Other successful GA applications have been found in robotic maneuvering, game theory, and of course, simulations of biological populations and learning.

Conventional Genetic Algorithms

Conventional GA's are based on the following model:

To begin with, a population of random binary strings, or organisms, is generated. Their fitness is evaluated, and in order to improve the population fitness as a whole, the more fit strings are given more copies in the mating pool. The strings are then mated by the following procedure:

Strings are divided into groups of two. In each group, a point along the length of a string is selected at random. The mating is done by exchanging the digits in the strings located after the selected point, forming two organisms that are different from their parents (unless the digits located after the mating point are the same in each parent). This is done to all of the pairs in the mating pool, and the resulting strings are mutated as follows:

A randomization done with the preselected mutation rate determines which, if any, of the binary digits in the resulting population will be mutated. Any selected digit is then reversed from 0 to 1, or vice versa. Mutation rates that are too high tend to depress the population fitness and stability, often destroying potentially successful organisms. If the rates are too low, the population tends to converge on one genotype, loosing its diversity and thus its ability to investigate new organisms that are potentially more fit. Studies of real-life populations indicate a mutation probability between 1 in 10,000 and 1 in 100,000 per gene (Mange, *Genetics: Human Aspects*, p.438). For GA's, this rate is usually increased by a magnitude of ten to between 1 in 1,000 and 1 in 10,000 genes mutated per generation.

A Sample Genetic Algorithm

The only thing that needs to be changed in this model to apply it to different applications is the fitness function. The fitness function *is* the application, because it tells each organism how well it is performing in the problem being investigated. The following example will clarify this by demonstrating how a GA works to find the

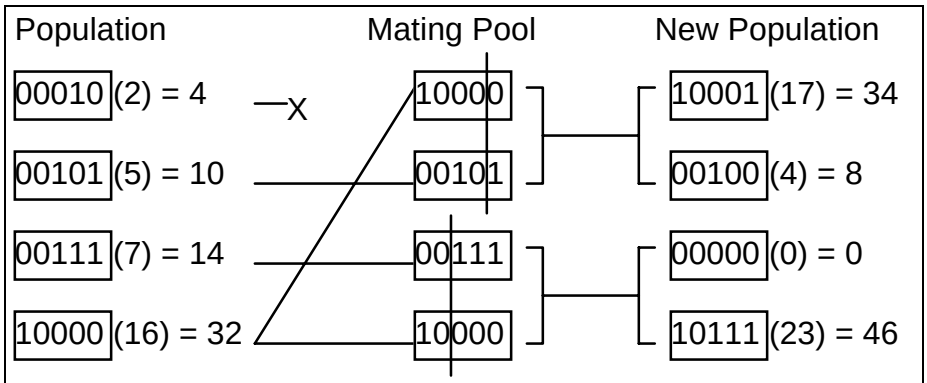


Figure 1: *A demonstration of a typical genetic algorithm on the function* y = 2x.

maximum value of a two-dimensional problem, in this case defined by the mathematical function $y = 2x$, a

straight line in which x represents the organism and y defines its fitness. (In other applications, the value of y may be determined by a computer simulation of a turbine, or robot moving along a test path.):

For this program, we decide to look only at the points on the x-axis less than 32, allowing us to use strings that are only five binary digits long ('11111' = 31). To begin with, four random strings are generated (see Figure 1), and their fitnesses are calculated. To determine which ones to add to the mating pool of four strings, we compare each organism's fitness to the sum of the fitnesses (60), multiply it by the number of mating positions (4), and round off the result. For example, with organism 16, this results in the equation $\frac{32}{60} \times 4 \approx 2.13 \Rightarrow 2$; and so organism 16 gets 2 copies of itself in the mating pool.

Mating is then done by randomly pairing the 4 organisms in the mating pool. The organisms are lined up, a random mating point is selected, and the digits after the point are swapped as shown. In this case, the new population not only has a higher maximum fitness (46), but also a higher total fitness (88) than the initial population. Mutation would be highly unlikely with a population this small, but just for demonstration, we could say that the fourth digit of organism 17 mutated to a 1, yielding an organism 19 with fitness 38.

A Recessive/Dominant Genetic Algorithm

In genetic terms, the conventional model just described is monoploid and does not provide for recessiveness in the genes (binary digits). My Recessive/Dominant genetic algorithm simulates diploid organisms. To simulate a population of organisms with recessiveness, I had to modify the conventional model in the following ways:

First, an organism is not represented by binary digits, but by genes made up of any combination of two of the following alleles: r0, d0, r1, d1--where r/d signifies recessiveness/dominance. To determine an organism's fitness, the genes are converted to a typical binary organism in a manner similar to the way recessive and dominant alleles interact in biological organisms to manifest traits--see Figure 2. (I follow the simplest such interaction, not allowing for such phenomena as intermediates, codominants, and overdominants. However, if this R/D model GA were ever applied to real-world situations, it is conceivable that such phenomena would be appropriate to consider, and would then be incorporated into the algorithm.)

r0r0	0
d0r0	0
d0r1	0
d0d0	0
d0d1	1
r0r1	0
r1r1	1
d1r1	1
d1r0	1
d1d1	1

Figure 2: *The key for translating R/D to binary.*

Organisms are copied to the gene pool by randomly selecting half of the alleles of an organism to put in one gamete, and then taking the other half for the other gamete rather than taking a copy of the whole organism, as in the conventional model. Mating is done by randomly combining gametes from the gene pool. Mutations are done on the gametes before combination.

The Tests

After writing C programs based on the two genetic algorithms described, I did two tests to compare my R/D model with the conventional one. This first one applies the GA's to a simulation of a biological population. To better simulate the realities of population survival, I did the following with both models: Rather than keeping the population fixed at an arbitrary number, I began with two random organisms. In subsequent generations, the number of organisms in the population was set equal to the highest fitness achieved in

the previous generation. This was done on the premise that as a population of organisms becomes more fit, or more adapted to its environment, it is better able to take advantage of the resources available to it and can thus support more organisms than a less fit population. For example, a population of ampicillin resistant bacteria will be far larger, in a dish of ampicillin, than a population of non-resistant bacteria.

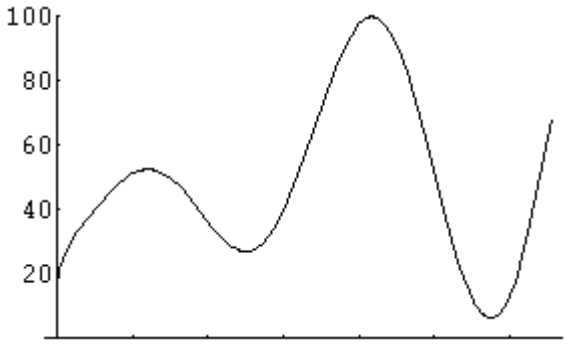
The fitness for all of the testing was determined by two equations designed to return fitness values between 1 and 100 (see Graphs A and B). First, the R/D model was run, each simulation started with two 16 gene organisms randomly generated, but with relatively low fitnesses. 4 simulations were performed on each graph, with each simulation running for 250 generations. Next, the starting organisms for the eight simulations were normalized by setting the gene for each manifest '1' to 'd1d1' and for each manifest '0' to 'd0d0'. Then the simulations were rerun on the R/D model using the normalized starting organisms. Finally, the manifest starting organisms were converted to binary and run using the conventional genetic algorithm.

The purpose of the normalizing was to provide a control for the R/D model. Giving it completely random starting organisms essentially gives it twice as much information, or area in the search space, to run with. Though only half of this information is present in the first generation, the recessive alleles quickly come out when mated with other gametes. This assumption is confirmed by the results, which show that the random R/D model is the quickest to locate a high fitness: The relatively high average *Max* (see Figure 3) implies that it found that maximum relatively early on--if it hadn't, the weight of a large number of early low maximums would have led to a much lower average *Max*. The normalized model, though it still contains the advantages that appear to be inherent in the R/D model, does not have the benefit of the additional latent search space that the random model has. Because of this, its results are between those of the random R/D and conventional models, though closer to the random one (see Figure 4).

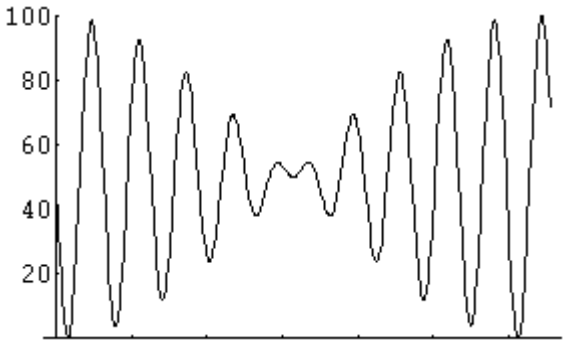
The *Avg* columns (Figure 3) indicate the average of each generation's average fitness. A higher average indicates a fit population throughout the test, meaning that the population as a whole was relatively quick to adapt, or else that the organisms that

		Random R/D Model			Normalized R/D Model			Conventional Model		
	Avg:	Max	Avg	Range	Max	Avg	Range	Max	Avg	Range
G	A	98.84	81.65	44.67	30.46	26.21	15.50	59.61	16.36	47.66
r		97.63	80.72	44.44	77.10	54.65	43.86	48.11	18.85	36.11
a		97.63	77.57	57.88	98.22	76.77	57.80	45.84	18.22	33.84
p		98.64	87.88	49.44	98.50	69.12	57.88	53.24	19.02	41.24
h	B	97.66	88.76	28.28	97.31	88.60	33.37	52.86	48.85	15.54
		75.96	57.52	72.09	94.76	72.16	58.12	66.50	48.95	28.98
		97.25	68.69	65.12	70.92	68.48	20.65	83.52	49.89	47.26
		97.20	85.59	44.10	87.90	61.82	43.85	91.82	51.84	57.17
	Avg	95.10	78.55	50.75	81.90	64.73	41.38	62.69	34.00	38.47

Figure 3: *Results of simulations on* Graphs A *and* B. *Each row represents a different test; all statistics given are the averages for the test, except for the last row, which is the average for the model.*



Graph A: $y = .001x \sin\left[\frac{x}{5200}\right] + 2x^{.3} + 10$



Graph B: $y = -50\cos\left[\frac{x}{20000}\right]\sin\left[\frac{x}{1000}\right] + 50$

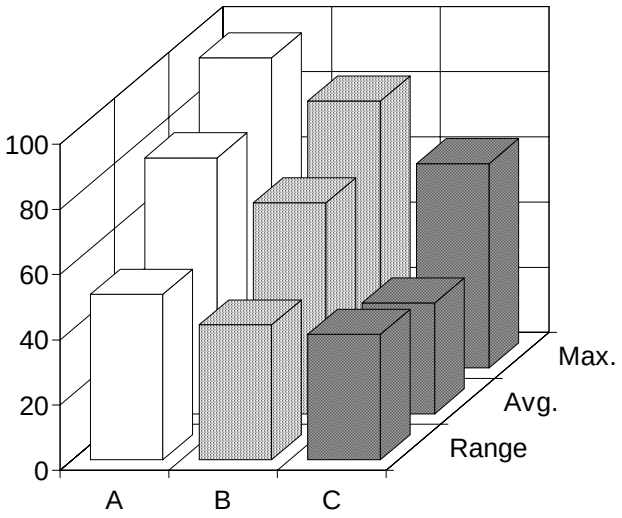


Figure 4: *The averages of the models;* A *is the* Random R/D, B *is the* Normalized R/D, *and* C *is the* Conventional *Model.*

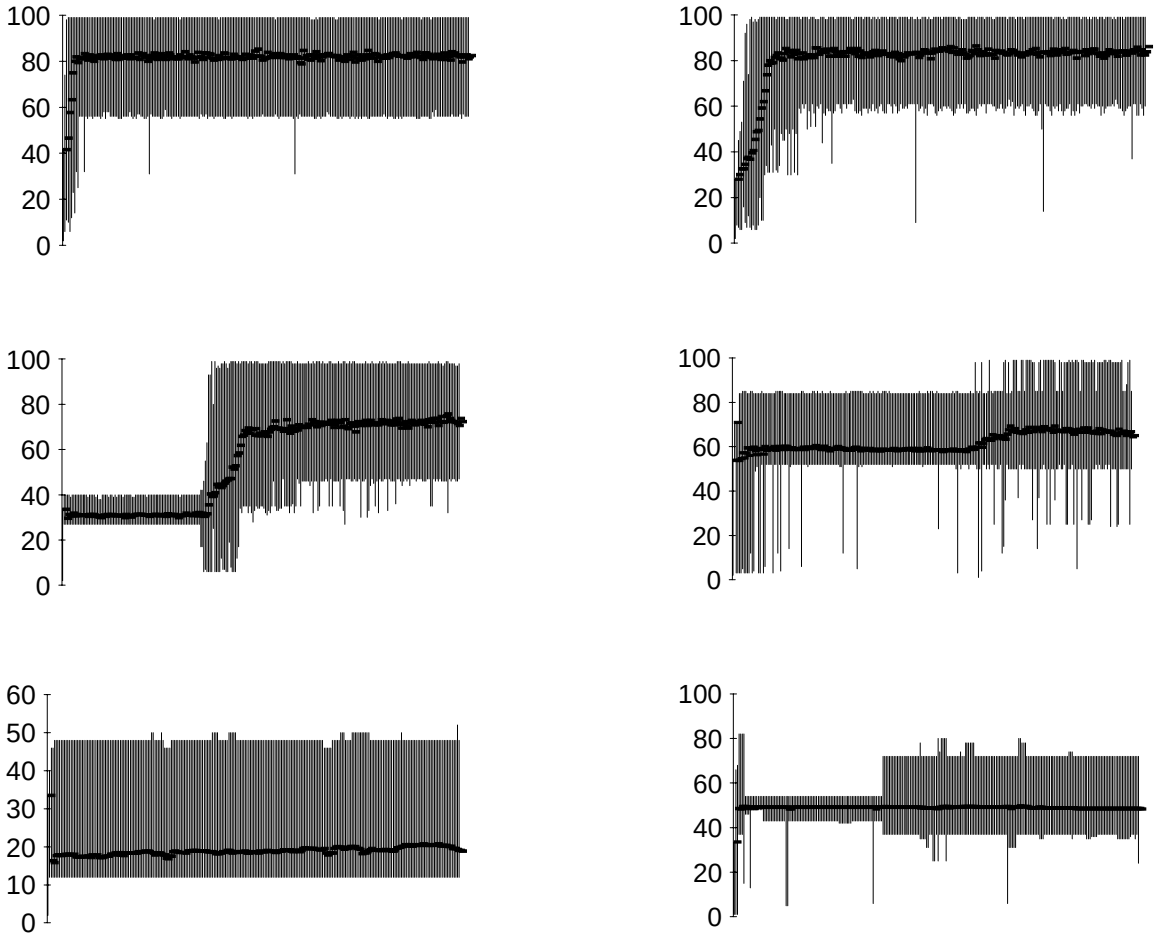


Figure 5: *Samples of genetic algorithm tests. The* y*-axis is the fitness, and each bar along the* x*-axis represents a generation. The top and bottom of each bar are the highest and lowest organisms of the generation; the tic between the two ends represents the average fitness for the generation. Each graph contains 250 generations. [First row contains the first and second tests of the* Random R/D *model (see* Figure 3*). Second row is the second and eighth tests of the* Normalized R/D *model. Third row is the second and sixth tests of the* Conventional *model.]*

adapted quickly also spread their genes through the population quickly. Clearly, the R/D models are superior to the conventional model in this respect.

The *Range*, or the average difference between the greatest and least fit organisms each generation, appears to indicate the search space investigated during each generation. An alternate interpretation of a high range could be a stagnant population, because if the bad genes are not replaced fast enough, or the good genes are not spread quickly, extreme lows would be remain in the population, and thus yield a higher range between themselves and the highs. However, a glance at a representative test (Figure 5) of the

R/D models, which exhibited the highest ranges, shows that this is not the case. The inconsistent lower spike seen in the charts in Figure 5 indicates that the population is reinvestigating areas, not remaining trapped in them. This is a tribute to the model's adaptability, and would prove especially useful if the model had to adapt to a changing fitness function, much as organisms must do in reality. On the other hand, the sample conventional test appears to be quite stagnant, probably only emerging from its convergences on small areas of fitness as a result of a mutation, not mating.

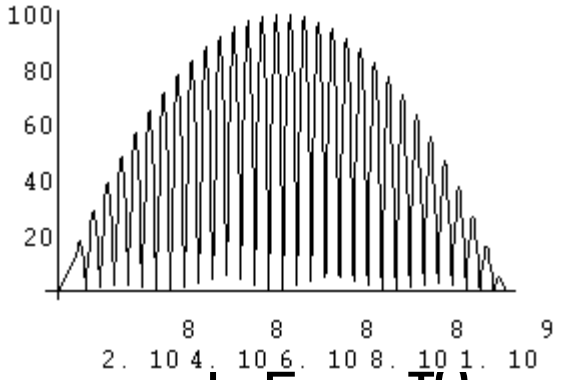
Another advantage the R/D has over the conventional model is its mating system. The conventional model suffers from a phenomenon called schemata instability. If, for example, a twenty-gene organism is only at its optimal if both the first and last genes are 1, it will suffer from schemata instability because the crossover during mating will always separate the 2 genes. If they are ever optimized, their children will not be, unless they happen to be mated with a good organism. Many systems have been proposed to solve this problem (e.g., schemata inversion and schemata reordering), but they cannot completely eliminate the phenomenon. Fortunately, the R/D model does not suffer this problem.

The second test was a simple comparison of the two models using the function shown in Graph C. The results (Figure 6) favor the R/D model--it surpasses conventional results that use twice as many organisms. It is true, however, that the R/D model takes more than 1.5 times as much computational power; but even if it took twice as long to execute, the intrinsic advantages of the algorithm leave it preferable to the conventional one. I am continually making changes to the program which have/will improve both its speed and performance further.

The power of genetic algorithms is incredible, and it increases with the difficulty of the problem. For example, I have run 100 organism tests searching over 10^{30} points that have found maximums to 3 decimal places in under 100 generations--far faster than a simple search method. While there is still much investigation to be done, it is clear where the promise lies.

#/Organisms Generation	GA	Avg Max/ Generation	Avg Max/ Test	Avg Avg/ Generation	Max's of Tests Below Key Levels
200	Cnv.	980.63	998.47	719.74	9 < 999; 2 < 997
		994.92	998.97	730.61	6 < 999; 0 < 997
		987.77	998.72	725.17	15 < 999; 2 < 997
	R/D	982.90	998.81	695.53	4 < 999; 1 < 997
		989.17	999.17	674.06	1 < 999; 1 < 997
		986.04	998.99	684.80	5 < 999; 2 < 997
100	Cnv.	981.97	997.31	784.00	12 < 999; 5 < 997
		985.56	997.96	747.77	10 < 999; 5 < 997
		983.77	997.64	765.89	22 < 999; 10 < 997
	R/D	977.73	999.03	730.69	5 < 999; 0 < 997
		974.78	998.70	708.09	4 < 999; 1 < 997
		976.26	998.87	719.39	9 < 999; 1 < 997
50	Cnv.	984.41	993.52	760.47	20 < 999; 13 < 997
		971.13	991.16	732.13	16 < 999; 12 < 997
		977.77	992.34	746.30	36 < 999; 25 < 997
	R/D	967.90	994.83	727.42	14 < 999; 11 < 997
		961.53	996.46	718.35	9 < 999; 8 < 997
		964.72	995.65	722.89	23 < 999; 19 < 997
20	Cnv.	969.55	991.68	751.18	16 < 999; 10 < 997
		962.97	986.05	747.53	16 < 999; 15 < 997
		966.26	988.86	749.35	32 < 999; 25 < 997
	R/D	964.18	994.97	756.81	15 < 999; 8 < 997
		949.98	995.37	721.05	14 < 999; 7 < 997
		957.08	995.17	738.93	29 < 999; 15 < 997

Figure 6: *Results of simulations on* Graph C. *The shaded boxes contain results of 20 tests of 50 generations each; the white boxes total all 40 tests on each model. The most important number, from an application standpoint, is the* Avg Max/Test*--the average of the highest fitnesses found in each test. The* Max's of Tests Below Key Levels *merely breaks that number down to give a better sense of what happened. I am unsure of why the* Avg. Max/Generation *was so poor for the R/D; it warrants further investigation.*



Graph C: $y = 25\left|\sin\left(\frac{x - 5\cdot 10^{8}}{2.5\cdot 10^{8}}\right)\sin\frac{x}{10^{7}}\right|$

Bibliography

Goldberg, David E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

Holland, John H. Genetic Algorithms. *Scientific American*, Vol. 267 #1, pp. 66-72.

Wayner, Peter. Genetic Algorithms. *Byte*, Vol. 16 #1, pp. 361-368.

Williams, John D. Designing With Genes. *Midnight Engineering*, Vol. 3 #5, pp. 29-33.

Note: After I finished the bulk of this report, I happened to be looking through some chapters of Goldberg which I hadn't yet read. Unfortunately, I discovered that diploid GA's have already been proposed and written. However, there is still hope because apparently none of them worked very well, or at least not well enough to overtake the conventional model. Mine does work better than the conventional model, and so while my idea isn't new, my genetic algorithm is.