CS411
Intro to Genetic Algorithms

A GA is a search algorithm based on the mechanics of natural selection and natural genetics. Using the idea of the survival of the fittest among string structures with a randomized information exchange, they form a powerful search algorithm.

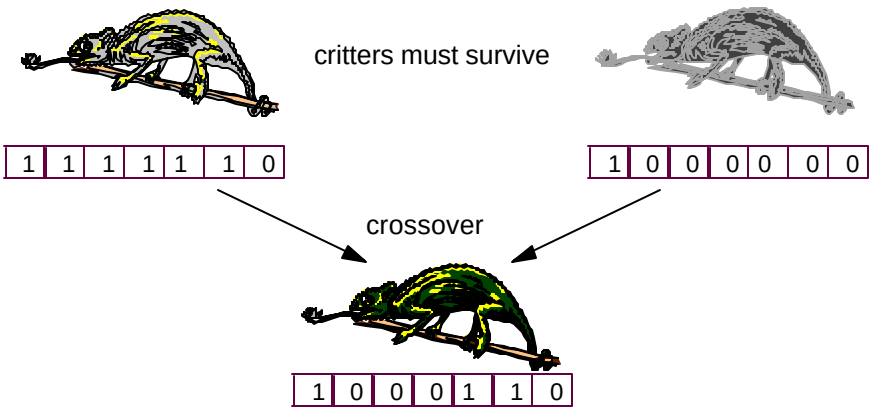
The idea was invented by John Holland at the University of Michigan in 1970's, but only in the last decade has it become popular, spawning its own conferences.

Let's say we have a population of critters in the world. How do they survive according to Darwinian evolution?

Population of critters
Some of them are better able to survive in their environment (smaller and require less food, bigger and stronger, longer neck to reach food)
Each critter has a measure of "fitness" - how well it survives in the environment
The more fit individuals will tend to survive
The less fit individuals will tend to die off
The individuals that survive will have sex and reproduce, passing their surviving characteristics on to their children through their genes
random crossover, get characteristics of both parents
random mutation, randomly changed genes
Presumably children will be even more fit than the previous generation
Repeat process, getting more and more fit individuals in each generation.

Dangers:
Need diverse genetic pool, or we can get inbreeding : stagnant population base
No guarantee that children will be better than parents, could be worse, could lose a super individual

Example: chameleon critters in the wild. Lets say that we show their genes as strings of bits:



In this example, we performed genetic crossover, getting a new chameleon that is even better than the first one.

Okay, so this may be the case, so what?

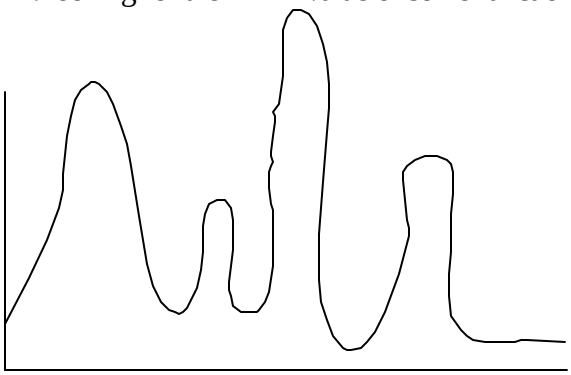
The important thing to note is that the population as a whole is learning. It is learning to adapt to the environment, defined by the fitness function. What if the fitness function is not how well something can survive in nature, but how well a particular program solves a computer problem? Then if the individuals are computer programs, and we have a known goal (fitness function), then we can evolve computer programs to solve the goal.

Population of critters	→	Population of computer solutions
Surviving in environment	→	Solving computer problem
Fitness measure in nature	→	Fitness measure solving computer problem
Fit individuals live, poor die	→	We play God and kill computer solutions that do poorly, keep those that do well. i.e. we are "breeding" the best solutions
Pass genes along via mating	→	Pass genes along through computer mating

Repeat process, getting more and more fit individuals in each generation.

Usually represent computer solutions as bit strings.

Simple Ex: looking for the MAX value of some function:



Traditional techniques: we could use hill climbing: look at neighbors, and go in direction of slope. Or we could use some calculus methods. One problem that may occur is the system might get stuck at *local minima*. Or we could go through all possibilities exhaustively, but if the search space is too large this doesn't work either.

Let's make our individuals just be numbers along the X axis, represented as bit strings, and initialize them randomly:

Individual 1	:	000000000
Individual 2	:	001010001
Individual 3	:	100110110
….		
Individual N	:	110101101

Fitness function: Y value of each solution. This is the fitness function. Note that even for NP complete problems, we can often compute a fitness (remember that solutions for NP Complete problems can be verified in Polynomial time).

Take the individuals that did the best, and probabilistically kill off the worst. Perform N matings so that we get a new population of size N:

Crossover: Randomly select crossover point, and swap code

Individual 1:	00101*0001*	
Individual 2:	*1001*10110	
New child :	100110001	; has characteristics of both parents, hopefully better than before

Or could have done:

Individual 1:	001010001	
Individual 2:	*100110110*	
New child:	100110111	; very little change in this case

Mutation: Just randomly flip some bits ; low probability of doing this

Individual:	011100101
New:	111100101

Mutation keeps the gene pool active and helps prevent stagnation.

Repeat process with each new generation until we get a satisfactory solution (or solution converges on some value).

This may look crazy, like it is equivalent to random search, but it does a very good job in many cases! It is much better than random search, and in many cases is better than many heuristic search algorithms. The randomization also helps us stay out of local minima, while still converging on a solution. Don't believe it works? Try it out! It's very simple to code.

In this simple example, we just used bit strings representations that mapped into numbers. They could just as easily have mapped into decision rules (IF-THEN…), probabilities for a bayesian classifier, linear discriminants, parameters for a neural network, or whatever else you would like. The overall genetic algorithm remains the same. We could even apply the technique to programs, so that we evolve pieces of code. This technique is called evolutionary programming, or genetic programming and was first proposed by John Koza of Stanford.