Genetic Algorithms and Genetic Programming

Pavia University and INFN
First Lecture: Genetic Algorithms

Eric W. Vaandering

ewv@fnal.gov

Vanderbilt University

Course Outline

- Machine learning techniques
- Genetic Algorithms
  - Correspondance with biological systems
  - Simple examples
  - Possible applications
- Genetic Programming
  - An example from HEP
  - Implementation

There will be three lectures and I'll be available to meet and discuss possible applications.

Machine Learning

There has been a long interest in teaching machines to "automatically" solve problems, given the broad parameters of the possible solutions.

For all but the simplest problems, an exhaustive or blind searches are impractical. (We'll see rather simple problems that can't be solved this way in the lifetime of the universe.)

There are numerous attempts to find these solutions: neural nets, simulated annealing, expert systems, etc.

To find the best solution, maybe we should take a clue from biology and the evolutionary process. ($\rightarrow$ Genetic Algorithms)

Since we will use computer programs to implement our solutions, maybe the *form* of our solution should *be* a computer program.

Combined, these last two points form the basis of

Genetic Programming

Genetic Algorithm: Definition

The Genetic Algorithm (GA) is a probabilistic search algorithm that iteratively transforms a set (population) of objects (usually a fi xed-length binary string), each with an associated fi tness value, into a new population of offspring objects using the Darwinian principle of natural selection and operations that mimic naturally occurring genetic operations, such as sexual recombination (crossover) and mutation.

Genetic Algorithms

These lectures deal mostly with Genetic Programming (GP). Genetic Algorithms are conceptually easier to understand, so I'll illustrate how the biological model applies to GA's before talking about GP. (Historically, GP was an outgrowth of GA.)

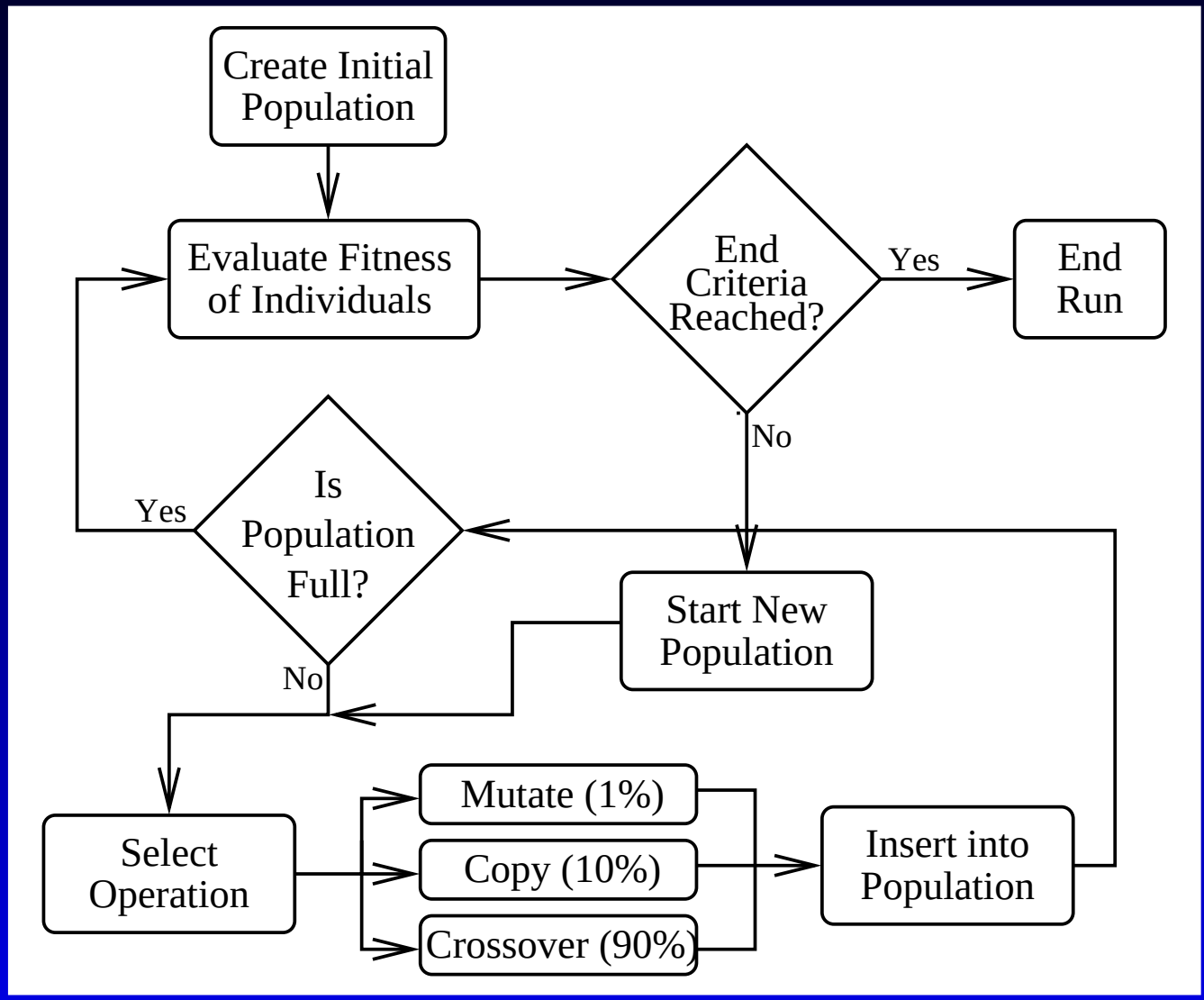
Begin with an analogy to human DNA:

- Double strand with four letter alphabet: G-A-T-C
  - A&T pair, G&C pair
  - Allows for error correction
- Fixed length strand of DNA contains many genes
- Each gene codes to one protein, contains many letters

Genetic algorithms

- Usually a two letter (binary) alphabet, single strand
- Fixed length strand of bits contains many genes
- Each gene codes to one part of the problem

GA Flowchart



The same general steps are used in Genetic Programming.

Populations

The Genetic Algorithm (GA) is a probabilistic search algorithm that iteratively transforms a set (population) of objects (usually a fi xed-length binary string), each with an associated fi tness value, into a new population of offspring objects using the Darwinian principle of natural selection and operations that mimic naturally occurring genetic operations, such as sexual recombination (crossover) and mutation.

Populations and Generations

Genetic algorithms work by transforming one group of individuals (typically a few hundred to a few thousand) in generation n into another group of individuals in generation $n + 1$.

Typically the number of individuals in each generation is the same. Usually no duplication is allowed in the 1^{st} (or 0^{th}) generation. Duplication *is* allowed in later generations. (Diversity decreases.)

There are GA implementations where change is not generational, but adiabatic. In these implementations, as a new individual is created, an old one is "killed," keeping the population size the same.

Representation

The Genetic Algorithm (GA) is a probabilistic search algorithm that iteratively transforms a set (population) of objects (usually a fi xed-length binary string), each with an associated fi tness value, into a new population of offspring objects using the Darwinian principle of natural selection and operations that mimic naturally occurring genetic operations, such as sexual recombination (crossover) and mutation.

Phenotype mapping

In the biological model, the genotype (the DNA) maps, via proteins, to the phenotype, the physical organism. Various genes manifest themselves as traits of the organism.

In a GA, the genes of the string maps to a solution of the problem. For instance in a damped simple harmonic oscillator function

$$x(t) = Ae^{-\beta t}\sin(\omega t + \phi)$$

A, β, ω, and ϕ could each be represented by a "gene" (maybe 32-bits each). The abstract string becomes a real manifestation.

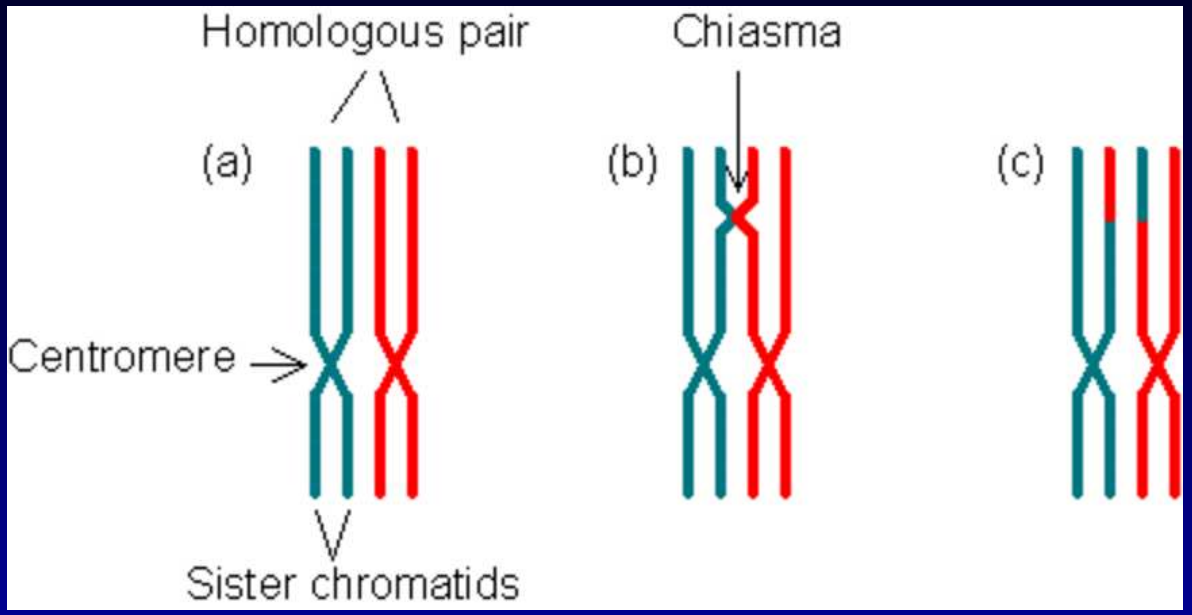
One can imagine any number of problems which can be solved with a series of weights or constants. State machine problems are also solvable this way.

Reproduction Methods

The Genetic Algorithm (GA) is a probabilistic search algorithm that iteratively transforms a set (population) of objects (usually a fi xed-length binary string), each with an associated fi tness value, into a new population of offspring objects using the Darwinian principle of natural selection and operations that mimic naturally occurring genetic operations, such as sexual recombination (crossover) and mutation.

Cross-over

Biological Model



Genetic algorithms:

- Choose two strings, pick point on strings
- Swap segments, create two new strings

000111011111100 ⟶ 000111 011111100

000101100010101 000101 100010101

gives two new children

000111100010101 and 000101011111100

Mutation

Mutations in nature change the genetic code for a small region of DNA. Usually are harmful or neutral; occasionally helpful (creates a better/different protein).

Mutations in GA are usually kept to a minimum and are single letter changes in the GA string.

This is one reason binary codes tend to be used: a flip in the least significant bits of a “gene” may not have a major effect on model.

E.g., a model with a 6-bit gene coding an integer:

$$101101 \rightarrow 101111$$
$$(45) \rightarrow (47)$$

A high mutation rate approximates a random search.

Mutations can restore lost (or never present) diversity.

Reproduction

Reproduction (or cloning) simply takes an individual from one generation and copies it into the next generation unchanged.

Some simple organisms reproduce this way.

This process preserves good genes at the expense of diversity.

One might think that there is a delicate balance between diversity and stability, but this doesn't seem to be the case when using these methods. (Extremes can mess things up, but lots of combinations seem to yield good results.)

Selection

The Genetic Algorithm (GA) is a probabilistic search algorithm that iteratively transforms a set (population) of objects (usually a fi xed-length binary string), each with an associated fi tness value, into a new population of offspring objects using the Darwinian principle of natural selection and operations that mimic naturally occurring genetic operations, such as sexual recombination (crossover) and mutation.

Survival of fittest

In nature, we know that the more fit an organism is for it's environment, the more likely it is to reproduce. This is one of the basic tenets of evolutionary theory.

- Organisms with serious deformities are still-born or die at a young age
- Faster, stronger, or longer lived organisms will produce more offspring

The Genetic Algorithm method mimics this by determining a *fitness* for each individual. Which individuals reproduce is based on that fitness.

- The better the fitness, the better the solution
- The problem *must* allow for inexact solutions. There may be a single *correct* solution, but there must be a way to distinguish between increasingly incorrect solutions. (Otherwise, we are again, engaging in a random search.)

Measuring Fitness

There are many ways to quantify the fitness of a model:

- Number of cases classified correctly vs. incorrectly
- Similarity of desired output to actual output (*e.g.* distance between two functions)
- Time to complete a task

Let's construct our problem such that better solutions have *lower* values (with perfect being 0). This is called the *standardized* fitness, f_s.

Let's define another quantity, the adjusted fitness:

$$f_a(i) = \frac{1}{1 + f_s(i)}$$

$f_a \approx 0$ for very unfit individuals, $= 1$ for perfect solution.
i denotes the number of the individual within the population.

Reproduction probabilities

Once we know $f_a(i)$ we can choose which solutions are allowed to reproduce such that better solutions are chosen more often.

The standard way of doing this is a weighting called "fitness proportionate." The probability of the i^{th} individual being selected for reproduction, $p(i)$, is

$$p(i) = \frac{f_a(i)}{\sum_j f_a(j)}$$

and of course $\sum_i p(i) = 1$.

Implications:

- The best individual is *most likely* to be chosen
- The best individual is *not guaranteed* to be chosen
- The worst individual *may* be chosen

Fitness Over selection

For complicated problems which require larger population sizes, “fitness-over-selection” is often used. In this method:

- Divide the population into two groups
  - a small group of “good” fitness
  - a larger group of “bad” fitness
- Most of the time, select an individual from the smaller group
- Rest of the time, select an individual from the larger group
- Within each group, use standard fitness rules

E.g., maybe the individuals which supply 32% of the total adjusted fitness will be chosen 80% of the time.

This tends to speed up evolution at the expense of diversity.

Tournament Selection

Another type of selection is also used to implement survival of the fittest. In tournament selection, a number of individuals (two or more) are selected randomly. The most fit from that group is selected to reproduce. The process is repeated to find a mating partner (if needed).

We see this behavior in nature too...

Tournament

Reward

Termination

How do we decide to terminate the evolution of a system?

In nature, of course, this never happens, evolution continues (responding to new pressures, maybe). Not acceptable for solving real world problems.

If a “perfect” solution is found, this is a good place to quit (what is a “perfect” solution)?

In cases where a perfect solution does not exist or is not found, we have to find other criteria.

Usually evolution will cease or slow considerably; this may be a good termination point. Maybe we have an upper limit on the amount of time that can be spent finding a solution.

Genetic Algorithm: Definition

The Genetic Algorithm (GA) is a probabilistic search algorithm that iteratively transforms a set (population) of objects (usually a fixed-length binary string), each with an associated fitness value, into a new population of offspring objects using the Darwinian principle of natural selection and operations that mimic naturally occurring genetic operations, such as sexual recombination (crossover) and mutation.

Probabilistic

Biological evolution is not purposeful or directed. There is no goal in mind, such as creating the human form.

How is a Genetic Algorithm probabilistic?

- Initial population is randomly generated
- Participants in reproduction randomly chosen
  - Best not guaranteed, worst not excluded
- Mutation and crossover points are determined randomly
- Fitness determination can be probabilistic too (Monte Carlo method)

Probabilistic

Biological evolution is not purposeful or directed. There is no goal in mind, such as creating the human form.

How is a Genetic Algorithm probabilistic?

- Initial population is randomly generated
- Participants in reproduction randomly chosen
  - Best not guaranteed, worst not excluded
- Mutation and crossover points are determined randomly
- Fitness determination can be probabilistic too (Monte Carlo method)

A probabilistic search can help avoid being stuck in local minima. This is a common problem with deterministic search algorithms.

An Easy Problem

A Difficult Problem

Fitness

Model

Probabilistic Searches

A probabilistic search is the only way to reliably ensure that you will find the true maximum or minimum. Other such methods include

- Beam searches
  - Randomly pick a point, then try to find maximum using a hill climbing technique
- Simulated annealing
  - Models reheating and cooling a metal; organizes into structures
  - Look for better solutions, but allow a worse solution if “temperature” is high

A Genetic Algorithm example

We've heard that an infinite number of monkeys with an infinite number of typewriters will eventually produce all of Shakespeare's (or Dante's) works. Let's try something a little simpler.

Let's look at how a genetic algorithm might be used to select the optimum solution: my initials "ewv" from all 3-letter words

Step 1 – Define a representation: a 15-bit string which will encode 3 letters (5 bits/letter).

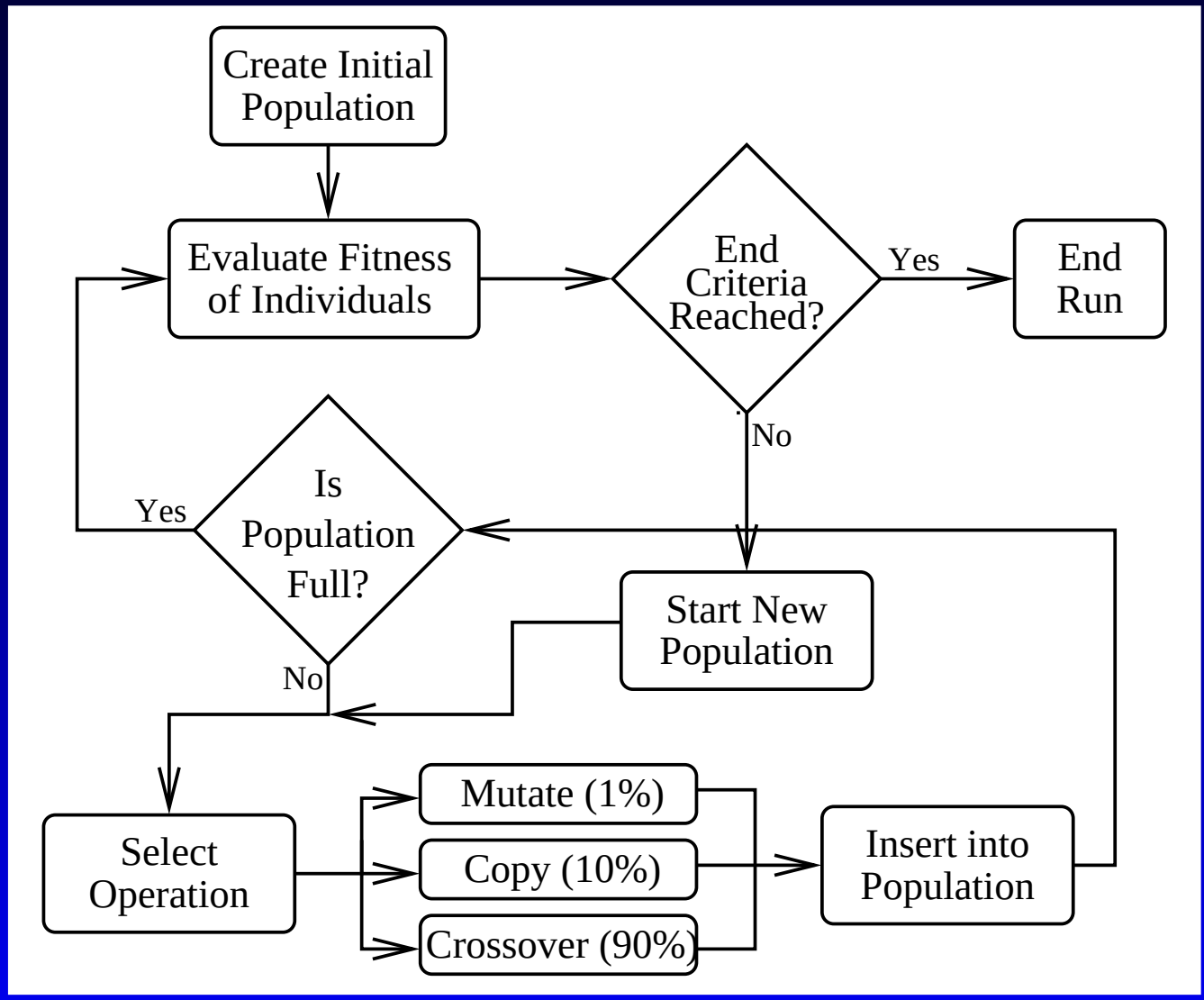
a = 1 = 00001	b = 2 = 00010	c = 3 = 00011
...	x = 25 = 11001	z = 26 = 11010

Step 2 – Define fitness: sum of how many steps away each letter is.

$$\text{aub} = |5 - 1| + |23 - 21| + |22 - 2| = 4 + 2 + 20 = 26$$

GA Flowchart

Let's recall the flowchart and where we are going with this:



GA example, continued

Step 3 – Generate a population of proposed solutions:

- Aside: There are $2^{15} \approx 33,000$ possible combinations. We are unlikely to randomly find the right one

In interest of time, look at 4 best from 100 generated (worst have fitness ~ 50)

Word	Binary	Fitness
fpy	00110 10000 11001	11
iyz	01001 11001 11010	10
bw?	00010 10111 11100	9
cxu	00011 11000 10101	4

Step 4 – Use natural selection to select mating pairs:

- Fitness of 5 selected $10\times$ as often as 50

Four new individuals in 2nd gen.

Step 5 – Pick a crossover point, two parents create two children

- Keep fixed length string

First crossover	Second crossover
00011 11000 10101 $(f = 4)$	00011 11000 10101 $(f = 4)$
00010 1 0111 11100 $(f = 9)$	01001 11001 11010 $(f = 10)$

Word	Binary	Fitness
cw?	00011 10111 11100	8
bxu	00010 11000 10101	5
bxv	00011 11000 10110	4
iyy	01001 11001 11001	9

Our four new combinations have better fitness than the original 100. In reality, we would generate 100 individuals in the 2nd generation using the original 100.

GA Example, continued

2^{nd} generation examples

Word	Binary	Fitness
cw?	00011 10111 11100	8
bxu	00010 11000 10101	5
bxv	00011 11000 10110	4
iyy	01001 11001 11001	9

The 3^{rd} generation might produce cwv (00011 11000 10110) which is *as close as we can get* in this simple example. This illustrates a few points:

- Diversity is important
- A perfect solution is not guaranteed
- Maybe we need mutation to restore lost bits
  - We lost the ability to generate a “1” in the 3rd position

A more complicated example

Generate “Pavia, Italia” from set of 90 characters. Same fitness as before. Solution space is $90^{13} = 2.5 \times 10^{25}$. It takes 36 generations of 2048 individuals (5.3×10^4) to find a perfect solution.

Best String	Fitness	Best String	Fitness	Best String	Fitness
ZtieK-/%cVjKd	183	Raskb.!Foagjf	30	Pavkb, Hsalia	5
COrlX&:Ooaf_h	113	Rask](Hvblkb	22	Pavkb, Hsalia	5
VOrlX&:Ooaf_h	106	Rask](Hvblkb	22	Pavkb, Hsalia	5
VOrlX&:Ooaf_h	106	R_rga)"Ivblkb	21	Pavka, Hsalia	4
W/mST0 UkbqfZ	97	R_rga)"Ivblkb	21	Pavha, Hsalia	3
PchY[”Igeqae	84	R‘vkb, Hsahhb	14	Pavha, Hsalia	3
MWvTb0 UkbqfZ	76	Pavkb, Hsahhb	11	Pavia, Hsalia	2
Sbrkr)"Qt[mWk	75	Pavkb, Hsahhb	11	Pavia, Hsalia	2
Pcp_/.!FoaWkb	58	Pavkb, Hsahhb	11	Pavia, Isalia	1
Rask](#HvYgjf	40	Pavkb, Hsahhb	11	Pavia, Isalia	1
Rasd]*"Orbgjf	38	Pavkb, Hsaljb	7	Pavia, Isalia	1
R_rga)"Joahhh	33	Pavkb, Hsaljb	7	Pavia, Italia	0

This model has a little bit of mutation.

How does the GA work?

Let’s go back to our “initial” example:

What is the reason ‘fpy’ matches ‘ewv’ pretty well?

- Is it that ‘f’ matches ‘e?’
- Is it that ‘p’ matches ‘w?’
- Is it that ‘y’ matches ‘v?’
- Is it some combination of these?

The Genetic Algorithm *doesn’t know* the answer. But it does know that ’fpy’ is better than ’zaa’ and it knows that ’fpy’ is better than average. So something must be “right” about ‘fpy.’

The same thing is true in biology. We might not know which combination of traits an organism might possess enable it to reproduce.

How does the GA work?

Of the combinations on the previous page, the GA wants to determine which ones are important.

Implicitly (but not explicitly) what the GA does is attribute success or failure to all these possibilities (and combinations).

- If one explanation is associated with both good and bad fitness, it may be unimportant.
- The GA figures this out automatically

In a sense, it does what you or I might do: change something and see what makes a difference. But, the GA does this *statistically* by varying many things at a time.

In practice, if there are important genes, they are usually "locked-in" quickly.

Some suggested applications

Now that we've seen how GAs work, what are some possible applications?

- "Fits" with a number of parameters, especially those with local minima that conventional fitting programs may become stuck in
- Weighting scheme to classify events (*e.g.* a spam filter)
- Calibrating a multi-channel piece of equipment (*e.g.*, a calorimeter)
- Many, many more

GA & GP Resources

There is a lot of information on the web about Genetic Algorithms and Programming:

- `http://www.aic.nrl.navy.mil/galist/` — Genetic Algorithms
- `http://www.genetic-programming.org/` — John Koza

Software frameworks for both GA and GP exist in almost every language (most have several)

- `http://www.genetic-programming.com/coursemainpage.html#_Sources_of_Software`
- `http://zhar.net/gnu-linux/howto/` – GNU AI HowTo (GA/GP/Neural nets, etc.)
- `http://www.grammatical-evolution.org` — GA–GP Translator