## Artificial Life Lecture 2

**Evolution and Genetic Algorithms**

The original definition of Artificial Life, by Langton and others, concentrated on what counted as a synthesis of (effectively) living artefacts, without regard to origins or evolution.

Despite this, very quickly a high proportion of Alife work came to be dependent on some form of **evolutionary** ideas.



## Biological Evolution

Read (strongly recommended, readable and fresh) the original C. Darwin 'On the Origin of Species'

Also John Maynard Smith 'The Theory of Evolution'

Richard Dawkins 'The Selfish Gene' etc.

M Ridley "Evolution" – (textbook)



## Evolution

The context of evolution is a **population** (of organisms, objects, agents ...) that survive for a limited time (usually) and then die. Some produce **offspring** for succeeding generations, the **'fitter'** ones tend to produce more.

Over many generations, the make-up of the population changes. Without the need for any individual to change, successive generations, the 'species' changes, in some sense (usually) adapts to the conditions.



## 3 Requirements

✓**HEREDITY** - offspring are (roughly) identical to their parents

✓**VARIABILITY** - except not exactly the same, some significant variation

✓**SELECTION** - the 'fitter' ones are likely to have more offspring



## Selection

Variability is usually **random** and **undirected**

Selection is usually **unrandom** and **directed**

In natural evolution the 'direction' of selection does not imply a **conscious Director** -- cf Blind Watchmaker.

In artificial evolution often **you** are the director.



## Neo-Darwinism

Darwin invented the theory of evolution without any modern notion of genetics - that waited until Mendel's contributions were recognised.

neo-Darwinian theory = Darwin + Mendel + some maths
(eg Fisher, Haldane, Sewall Wright)

## (Artificial) Genetics

In Genetic Algorithm (GA) terminology, the **genotype** is the full set of genes that any individual in the population has.

The **phenotype** is the individual potential solution to the problem, that the genotype **'encodes'.**

So if you are evolving with a GA the control structure, the 'nervous system' of a robot, then the genotype could be a string of 0s and 1s 00101001001110010100 1 and the phenotype would be the actual architecture of the control system which this genotype encoded.



## Possible examples …

You could be evolving for optimal timetables

*genotype: string listing room/student allocations*

*fitness: (negative) number of clashes*

Or for optimal aircraft wing design

*genotype: string listing various wing dimensions*

*fitness: formula based on lift/drag/cost of wing*

Or for … … … …



## An example

It is up to **you** to design an appropriate encoding system.

Eg. Evolving paper gliders

| | |
|---|---|
| Fold TL to BR towards you<br>Fold horiz middle away<br>Fold vertical middle towards | Fold TR to BL towards you<br>Fold horiz middle away<br>Fold vertical middle away |

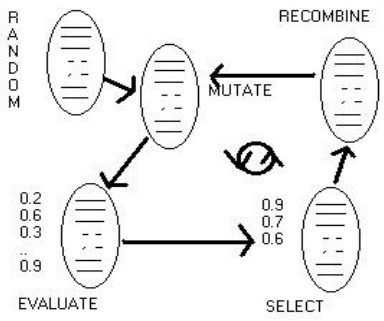


## Evolving paper gliders

1. Generate 20 random sequences of folding instructions
2. Fold each piece of paper according to instructions written on them
3. Throw them all out of the window
4. Pick up the ones that went furthest, look at the instrns
5. Produce 20 new pieces of paper, writing on each bits of sequences from parent pieces of paper
6. Repeat from (2) on.

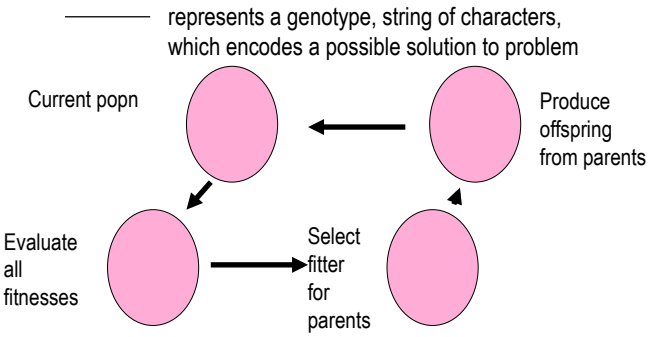


## Basic Genetic Algorithm



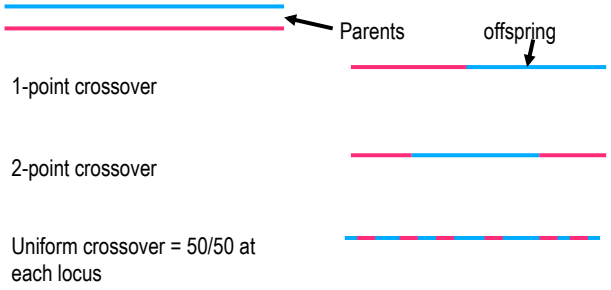



## Basic GA -- continued



## Recombination

Typically 2 parents recombine to produce an offspring






## Mutation

**After** an offspring has been produced from two parents (if sexual GA) or from one parent (if asexual GA)



**Mutate** at randomly chosen loci with some probability



**Locus** = a position on the genotype



## A trivial example

Max-Ones – you have to find the string of 10 bits that matches as closely as possible this string: 1111111111

… and yes, clearly the answer will be 1111111111, but pretend that you don't know this. A fitness function:-

```
int evaluate(int *g) {
        int i,r=0;
        for (i=0;i<10;i++) r += (g(i) == 1);
        return(r);
}
```



## Program structure

Initialise a population of (eg) 30 strings of length 10

```
int popn[30][10];
void initialise_popn() {
        int i,j;
        for (i=0;i<30;i++)
                for (j=0;j<10;j++)
                        popn[i][j]= flip_a_bit();
}
```



## Main Program Loop

```
For n times round generation loop
        evaluate all the population (of 30)
        select preferentially the fitter ones as parents
        for 30 times round repro loop
                pick 2 from parental pool
                recombine to make 1 offspring
                mutate the offspring
        end repro loop
        throw away parental generation and replace with offspring
End generation loop
```



## Variant GA methods

We have already mentioned different recombination (crossover) methods - 1-pt, 2-pt, uniform.

You can have a GA with no recombination -- asexual with mutation only.

Mutation rates can be varied, with effects on how well the GA works.

Population size -- big or small? (Often 30 - 100)

## Selection Methods

Eg. **Truncation Selection**.
All parents come from top-scoring 50% (or 20% or ..)

A different common method: **Fitness-proportionate**
If fitnesses of (an example small) population are
2 and 5 and 3 and 7 and 4 total 21
then to generate each offspring you select mum with
2/21 5/21 3/21 7/21 4/21 probability
and likewise to select dad. Repeat for each offspring.



## Different Selection Methods

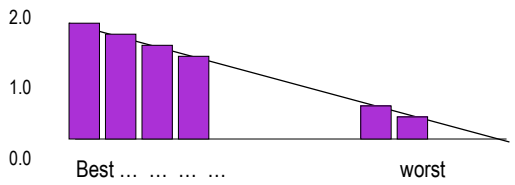
Problems with fitness-proportionate:
✓**How about** negative scores ?
✓**How about** if early on all scores are zero (or near-zero) bar one slightly bigger -- then it will 'unfairly' dominate the parenting of next generation?
✓**How about** if later on in GA all the scores vary slightly about some average (eg 1020, 1010, 1025, 1017 ...) then there will be very little selection pressure to improve through these small differences?

You will see in literature reference to scaling (eg sigma-scaling) to get around these problems.



## Rank Selection

With linear rank selection you line up all in the population according to rank, and give them probabilities of being selected-as-parent in proportion:






## More Rank Selection

Note with linear rank selection you ignore the absolute differences in scores, focus purely on ranking.

The 'line' in linear ranking need not slope from 2.0 to 0.0, it could eg slope from 1.5 to 0.5.
You could have non-linear ranking. But the most common way (I recommend unless you have good reasons otherwise) is linear slope from 2.0 to 0.0 as shown.

This means that the **best** can expect to have **twice** as many offspring as the **average.** Even below-average have a sporting chance of being parents.



## Elitism

Many people swear by elitism (...I don't!)

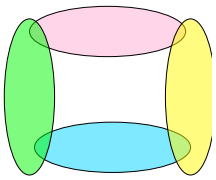
Elitism is the GA strategy whereby **as well as** producing the next generation through whichever selection, recombination, mutation methods you wish, you also **force** the direct unmutated copy of best-of-last generation into this generation -- 'never lose the best'.



## Demes / Geographical breeding

Brief mention here (more later):

One population quickly becomes fairly genetically converged, and thereafter tends to search just 'one corner' of the search space.



Multiple populations may search multiple corners – but be different evolutionary runs.

Maybe you can get the best of both worlds by having multiple sub-populations, with some form of **limited** breeding between.

## Genotype Encoding

Often genotypes in GA problems have **discrete** characters from a **finite** alphabet at each locus.

Eg. 0s and 1s for a binary genotype 010011001110
-- a bit like real DNA which has 4 characters GCAT

These often make sense with simple encodings of strategies, or connectivity matrices, or …



## Coding for Real Numbers

But sometimes you want to solve a problem with **real** numbers -- where a solution may include 3.14159

Obvious solution 1: binary encoding in a suitable number of bits. For 8-bit accuracy, specify max and min possible values of the variable to be coded. Divide this range by 256 points.

Then genes 00000000 to 11111111 can be decoded as 8-bit numbers, interpolated into this range.



## Coding for Many Real numbers

For eg 10 such real-valued variables, stick 10 such genes together into a genotype 80 bits long.
You may only need 4-bit or 6-bit accuracy, or whatever is appropriate to your problem.

A problem with binary encoding is that of '**Hamming cliffs**'

An 8-bit binary gene 01111111 encodes the next value to 10000000 -- yet despite being close in real values, these genes lie 8 mutations apart (a Hamming distance of 8 bits)



## Gray Coding

This is a 1-1 mapping which means that any 2 adjoining numbers are encoded by genes only 1 mutation apart (tho note reverse is not true!) -- no Hamming Cliffs

Rule of thumb to translate binary to Gray:
Start from left, copy the first bit,
thereafter when digit changes write 1
otherwise write 0.

Example with 3 bit numbers :--

| Bin | Actual | Gray |
|---|---|---|
| 000 | 0 | 000 |
| 001 | 1 | 001 |
| 010 | 2 | 011 |
| 011 | 3 | 010 |
| 100 | 4 | 110 |
| 101 | 5 | 111 |
| 110 | 6 | 101 |
| 111 | 7 | 100 |



## Other Evolutionary Algorithms

Note that GAs are just one type of evolutionary algorithm, and possibly not the best for particular purposes, including for encoding real numbers.

GAs were invented by John Holland around 1960s
Others you will come across include:

EP Evolutionary Programming
originally Fogel Owens and Walsh,
now David Fogel = Fogel Jr.



## And more …

ES Evolution Strategies invented in Germany
Rechenberg, Hans-Paul Schwefel
Especially for optimisation, real numbers

GP Genetic Programming
Developed by John Koza
(earlier version by N Cramer).

Evolving programs, usually Lisp-like, wide publicity.

## Which is best ?

Is there a universal algorithm ideal for all problems
-- **NO** !!

(cf 'No Free Lunch Theorem, Wolpert and MacReady)

Are some algorithms suitable for some problems
-- **PROBABLY YES**.

Is this a bit of a Black Art, aided by gossip as to what has worked well for other people -- **YES!**



## Recommendation …

For Design Problems, encoding discrete symbols eg binary, rather than reals, **my own initial heuristic** is:

GA (usually steady state rather than generational…)
selection: linear rank based, slope 2.0 to 0.0
sexual, uniform recombination
mutation rate very approx 1 mutation per (non-junk part of) genotype
Elitism not necessary
population size 30 - 100
In fact I ***always*** use a version of the Microbial one-liner GA (Lec 3)

**But others will disagree...**



## Sources of Information

David Goldberg 1989 "Genetic Algorithms in Search, Optimization and Machine Learning" Addison Wesley

Melanie Mitchell and Stephanie Forrest "Genetic Algorithms and Artificial Life".
Artificial Life v1 no3 pp 267-289, 1994.

Melanie Mitchell "An Intro to GAs" MIT Press 1998

Z Michalewicz "GAs + Data Structures = Evolution Programs" Springer Verlag 1996



## More …

plus many many more sources eg…

news group comp.ai.genetic

Be aware that there are many different opinions – and a lot of ill-informed nonsense.

Make sure that you distinguish GAs from EP ES GP.



## Advance Notice: Next Lecture

Tue Oct 12th: Lecture 3

"More on Evolutionary Algorithms", including the Microbial GA in one line of code.



## General Stuff

Information on lectures and seminars here:

http://www.cogs.susx.ac.uk/users/inmanh/easy/alife10/index.html

Check out details for this week's seminars

Alergic talks, typically Wed 16:30 in ARUN-401, first one this week --- opening session, new and existing researchers

Other discussion groups: Life and Mind, etc … …

COGS talks: