Non-Symbolic AI lecture 3

More on Evolutionary Algorithms

Last lecture discussed encoding **real** numbers as **bits** on a genotype (either binary encoding or Gray coding)

Sometimes people choose to have real numbers directly represented on the genotype -- which might be:

2.034 -30.678 0.005 102.567 ... ... -89.432

Recombination will work in the same way as with normal discretely encoded genotypes, but **mutations** will be hand **differently**

Mutating Real numbers

Various possibilities for **mutating** real numbers

One possibility is to change any mutated locus to a randomly chosen real number within the appropriate rang but this is very **disruptive.**

So more usually a form of '**creep mutation**' is used:
eg. add a random number in range [-0.1 +0.1]
or add a random number drawn from a **Gaussian** distribut
with mean zero, and appropriate range.

Evolution Strategies

If the problem you are tackling has all the parameters naturally expressed as real numbers, then maybe you should investigate Evolution Strategies
(see previous lecture)

These work primarily with a version of 'creep mutation', and this evolutionary paradigm has developed sophisticated strategies for **modifying** the amounts of 'creep' in different dimensions as evolution proceeds.

Different Search Algorithms

Or indeed you could look at **Simulated Annealing**
-- a non-evolutionary technique which nevertheless has sc
similarities.

These are all techniques for Search within a
many-dimensional, **real-valued** Search Space.

Genetic Algorithms may be more appropriate for Search
within high-dimensional **Discrete** Search Spaces.

Many *design* problems are such – but many are not.

Why Should GAs work ?

John Holland (1975) 'Adaptation in Natural and Artificial Systems' -- and most of the textbooks -- explain this with **Schema Theorem**, and ideas of **building blocks.**

Roughly speaking, building blocks are segments of the genotype which encode for functional components of the 'phenotype', or potential solution to the problem.

These building blocks can, in principle, be evaluated independently of all the rest, as varying between 'good' an 'bad'

Cartoon Version

Cartoon version of genotypes:

*** long legs *****************short arms**********

short legs************** long arms **********

^

Recombination (when crossover happens to land appropriately) allows different parents like these **in one generation** to produce a child with long legs **and** long arm

Schemata

Schemata (plural of schema) are a formalisation of this ide
a building block.

Consider binary genotypes of length 16. Let # be a **'wild-card'** or **'dont-care'** character.

Then ######00#010#####

is a schema of **order** 5 (5 specified alleles) and
of **defining length** 6 (length of segment which includes
specified alleles)

‘Processing Schemata’

Considering this schema
######00#010######
then
0000000001010000
is just **one** of many genotypes corresponding to this sche
-- and actually this genotype also corresponds simultaneou
to **many** other schemata.

Implicitly, the GA **'evaluates'** and **'processes'** loads
of schemata **in parallel**, every generation.

The Schema Theorem claims ...

The Schema Theorem claims ...

... that **schemata** of **short defining lengths** (coding for building blocks such as 'cartoon legs') will,

✓**IF** they are of above-average fitness, (..that is, evaluated whatever the other loci outside the schema are)

✓get **exponentially** increasing numbers of trials in successive generations.

Ie, **despite** recombination and mutation being **'disruptive'** (tho not too disruptive of short schemata)
'good building blocks' will multiply and **take over** ---
and **'mix and match'** with other 'good building blocks'

Implications of the Schema Theorem ??

The Schema Theorem is **formally** proved subject to certai
conditions.

This Theorem is widely **interpreted** as implying that **RECOMBINATION** is the **'powerhouse'** of GAs,

-- whereas mutation is just a 'background operator‘ (whose
only role is to add variety in loci where, throughout the wh
population, no variety is left).

Doubts about the Schema Theorem

The Schema Theorem is formally **correct.**

But nowadays many people (including myself) believe it ha
been **misinterpreted.**

The 'subject to certain conditions' bit means that this
exponential increase is only guaranteed over 1 generatio

-- thereafter the conditions **change!**

Recombination versus Mutation ?

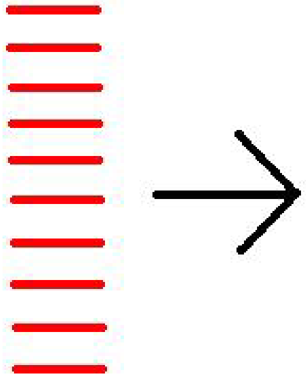
So be aware that despite this common view in the textboo
some people think that in some sense **MUTATION** is the
powerhouse of GAs, with recombination as a background
(tho often useful) genetic operator.

"The Schema Theorem is true, but not very significant"

Nevertheless, the common view of the importance of recombination lies behind the exclusive emphasis (often without any mutation) on recombination in
GP = Genetic Programming

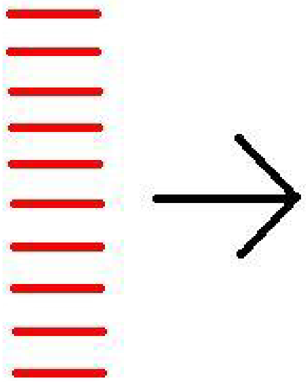
Some more GA wrinkles

You need not have a **generational** GA (where the whole population is swept aside every generation, and replaced by a fresh lot of offspring).



You can have a **STEADY STATE** GA.

Here just **ONE** member of the population is replaced at each time step, by the offspring of some others.



Steady State GA

Eg with a popn of 100:

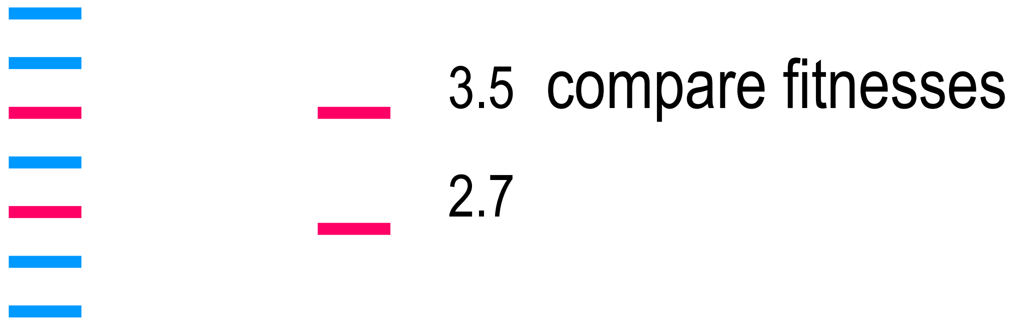
- ✓Choose a mum by some selection mechanism biased towards the fitter.
- ✓Choose a dad by same method.
- ✓Generate a child by recombination + mutation
- ✓Add the child to the population
- ✓Keep the numbers down to 100 by choosing someone else to die
(eg at random, or biased towards the less fit)

Roughly speaking, 100 times round this loop is equivalent to one generation of a generational GA

Tournament Selection

Here is a very simple way to implement the equivalent of linear rank selection in a Steady State GA

Pick 2 at random



Fittest of tournament is mum – choose dad the same way

Generate offspring from mum and dad – the new offspring replaces someone chosen at random.

Note: everyone else remains, including mum and dad.

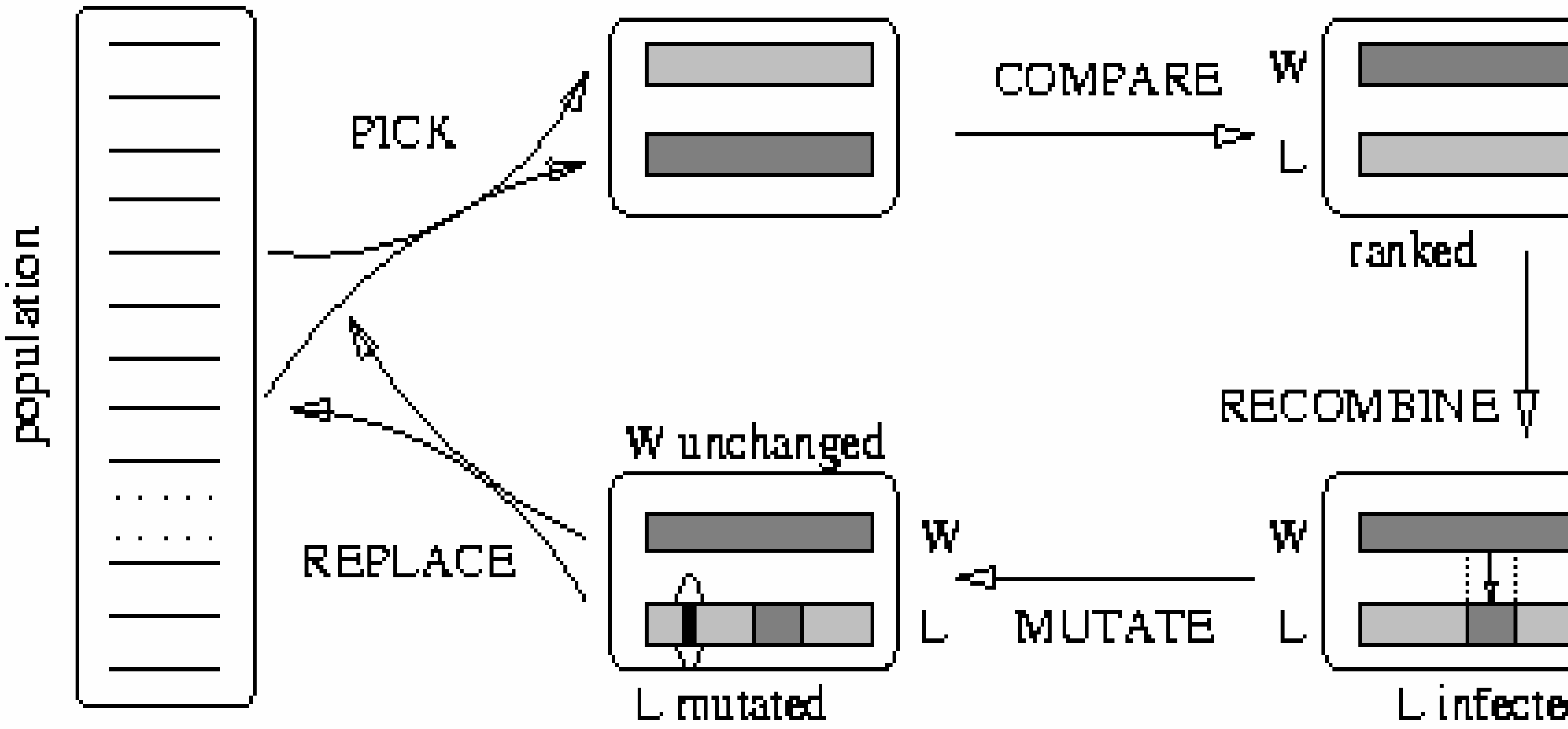
You needn’t even have death !

microbes can evolve by **horizontal** transmission of genes (within the same generation) rather than (or as well as) **vertical** transmission (down the generations, from parents to offspring).

ie recombination happens **within** generations

Microbial sex =
'hey, wanna swap some of my genes for yours?'
rather than
'lets make babies'

Microbial Genetic Algorithm – the picture



Microbial Genetic Algorithm – the algorithm

- ✓Pick two genotypes at random
- ✓Compare scores -> Winner and Loser
- ✓Go along genotype, at each locus
  - with some prob copy from Winner to Loser (overwrite)
  - with some prob mutate that locus of the Loser

So **ONLY** the Loser gets changed
(gives a version of Elitism for free!)

This allows what is technically a one-liner GA (bar the
evaluate() which is problem-specific) -- quite a long line !

Microbial Genetic Algorithm – the one-line

```
/* tournament loop */
for (t=0;t<END;t++)
        /*  loop along genotype of winner of tournament,
         selected in initial loop conditions */
        for (W=(evaluate(a=POP*drand48())>
                evaluate(b=POP*drand48()) ? a : b),
                        L=(W==a ? b : a), i=0; i<LEN; i++)
                /* throw dice to decide: cross or mutate */
                if ((r=drand48())<REC+MUT)
                /* update genotype of loser */
                gene[L][i]=(r<REC ? gene[W][i] : gene[L][i]^1);
```

… or slightly longer

```
int gene[POP][LEN];

Initialise genes at random;  define problem-specific evaluate(n)

/* tournament loop */
for (t=0;t<END;t++) {
        /*  pick 2 at random, find Winner and Loser */
        a=POP*drand48();
        do {b=POP*drand48()}
                while (a==b);  /*make sure a and b different */
        if (evaluate(a) > evaluate(b)) {W=a; L=b;}
        else {W=b; L=a;}
```

To be continued

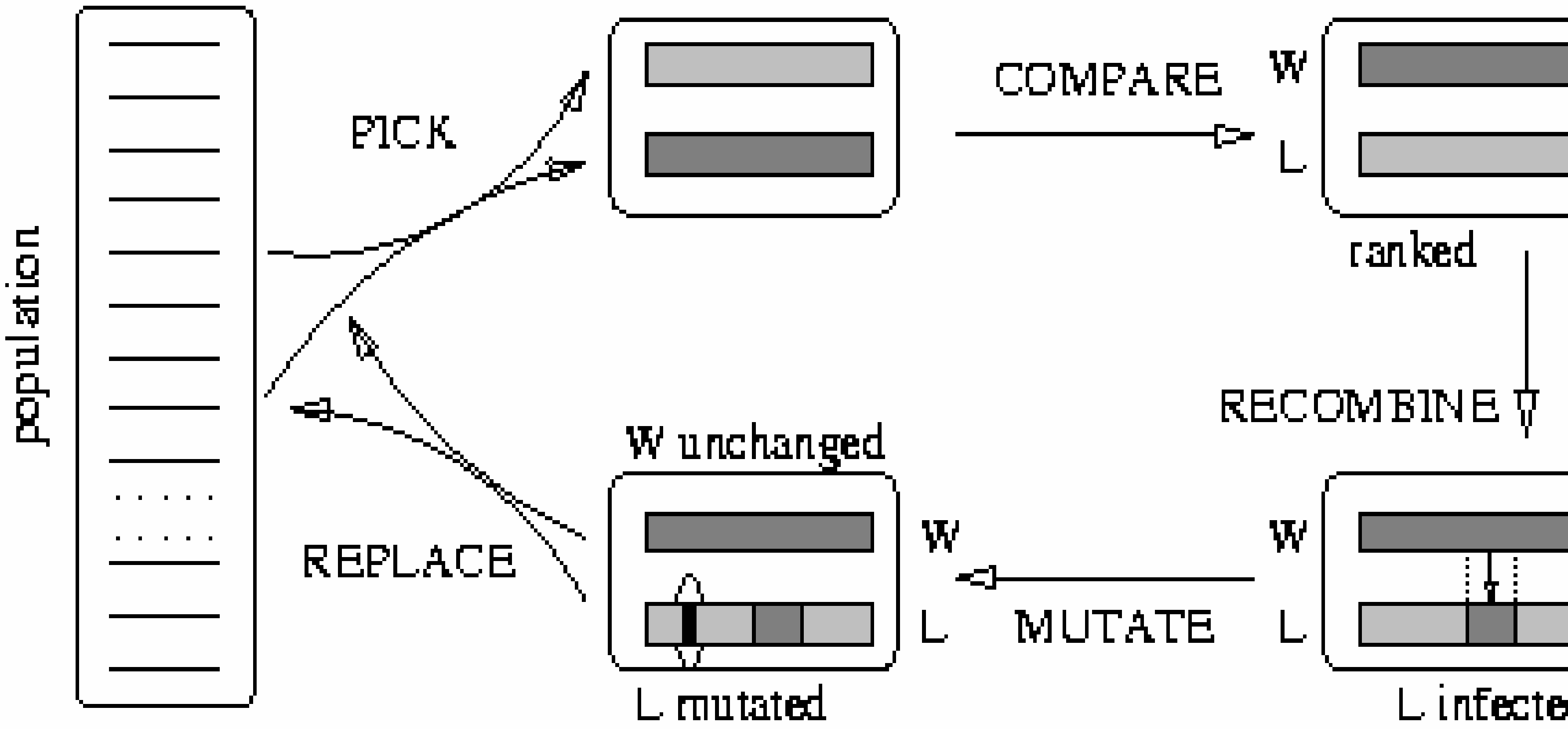
… continued

Continued …

```
        for (i=0;i<LEN;i++) {
                if (drand48()<REC) /* cross with probability REC */
                        gene[L][i]=gene[W][i];
                if (drand48()<MUT) /* mutate with probability MUT */
                        gene[L][i]=1-gene[L][i]; /* flip bit */
}       /* end tournament loop */
```

Possible values for **REC=0.5; ?** And **MUT=1.0/LEN; ?**

Microbial Genetic Algorithm – the picture



Is there a point ?

Microbial GA paper on my home page
http://www.cogs.susx.ac.uk/users/inmanh

It does actually work.

By no means guaranteed to be better than other GAs -- bu
does show **how really simple a GA can be**, and still work

Apart from the one line, it needs declaration of
gene[**POP**][**LEN**], initialisation of a random popn, and
evaluate(n) that returns fitness of nth member.

Embodied Evolution

Richard Watson, at Brandeis (papers available on web) ha
modified this to use with real robots in
'**Embodied Evolution**'.

Robots go around 'broadcasting' their genes, and listening
to other broadcasts.

Fitter robots 'shout louder' (or more often)

Weaker robots are more likely to listen in, and use the ger
they 'hear' to copy over their own

A mini-GA project

You have 10 cards numbered 1 to 10.

You have to divide them into 2 piles so that:

1) The **sum** of the first pile is as close as possible to 36
2) **And** the **product** of all in second pile is as close as poss to 3

Hint: call the piles **‘0’** and **‘’1’**, and use binary genotypes of length 10 to encode any possible solution.

[illegible]

Seminars/Lab classes Week 2

Week 2: Sessions Mon 14:00 and Thu 09:00 will be devoted to getting you *all* able to program this mini-GA project (and the experienced programmers can go further).

You should *all* turn up to your seminar slot with a legible printout on paper of your attempt at this, and with pens.

Preferably a red pen as well as a blue pen! As at some stage in the seminar your efforts will passed around, and you will be asked to assess and comment on each others work.

Any questions so far … ?

… with luck there should be time left for GA-related questi

Remember: details of seminars, copies of lectures etc are available on

http://www.informatics.susx.ac.uk/users/inmanh/non-symb