EVOLUTIONARY COMPUTING

Metaphors from biology

- Nature is good at evolving robust agents.
- Can we borrow such mechanisms to build artifical agents?
- It turns out that we can.
- We will look at two models:
  - Genetic algorithms
  - Genetic programming

Genetic algorithms

- The basic approach is:

```
genetic-algorithm(population,fitness)
{
  repeat
  {
   parents := selection(population,fitness)
   population := reproduction(parents)
  }
  until(enough fit individuals)
  return(fittest individual)
}
```

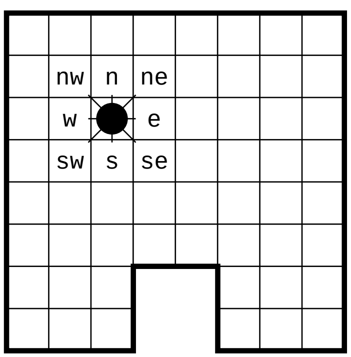

- This is *just* a fancy way of doing search.
- We code some part of the agent (e.g. action selection function) and decide how to do:
  - selection; and
  - reproduction.

  on it.
- When we have a bunch of individuals (as we typically do), each individual represents a state in the state-space of possible individuals.
- Establishing and evaluating a population is a (massively) parallel search though this space.

- To use the approach we have to instantiate:
  - What is the fitness function?
  - How is an individual represented?
  - How are individuals selected?
  - How do individuals reproduce?
- While these are to some extent domain dependent, we will look at some typical ways of doing this.

Fitness function

- The fitness function is the most domain dependent item.
- It is a function that takes an individual as an argument and returns a real number.
- In the example of our wall following robot a function could be:
  - The average number of moves out of *n* for which the robot makes the right action selection.
  - The average number of moves out of *n* for which the robot is adjacent to the boundary.
- Fitness functions often take time to evaluate.



Representation

- The classic representation is one in which features are coded as a binary "chromosome".
- (i.e. we code a sequence like 01110110 rather than AATGTCAT.)
- In our robot example, we could code up the action representation as a list of condition/action pairs:
  - One possible combination of sensor readings; followed by
  - The appropriate action.
- Sensor readings could be strings $n, ne, \ldots, nw$.
- Actions could be two digit binary numbers, 00 = north etc.

Selection

- Selection is usually a two stage process.
- First we limit the population:
  - Cull unfit indivduals to limit the population size.
- Then we select individuals to breed:
  - Random selection weighted towards fit individuals;
  - With replacement (so very fit individuals can breed several times).

Reproduction

- Two basic parts to reproduction:
  - Crossover; and
  - Mutation.
- First take two parents P1 and P2, and pick a number n between 1 and N = length of "chromosome".
- Create two "children", C1 and C2.
- C1 is the first n bits of P1 and the last $N - n$ bits of P2.
- C2 is the first n bits of P2 and the last $N - n$ bits of P1.

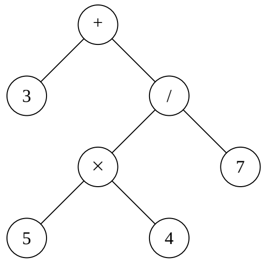
- Cross-over is analagous to state-space transitions in state-space search.
- Taking fit individuals and combining their features is a form of best-first search.
- It makes small "hill climbing" steps up the fitness function.
- However it can get stuck in local maxima.
- Mutation is a way of "jumping" to new areas of search space.
- We "mutate" random bits by flipping them.

- Again we have a lot of possible parameters to play with:
  - Fitness rating;
  - Selection probability;
  - Mutation rate;
  - Crossover point;
  - etc.
- As ever it is a black art choosing what these should be…
- "neural networks are the second-best way of doing just about anything, and genetic algorithms are the third" (Russell and Norvig).

Genetic programming

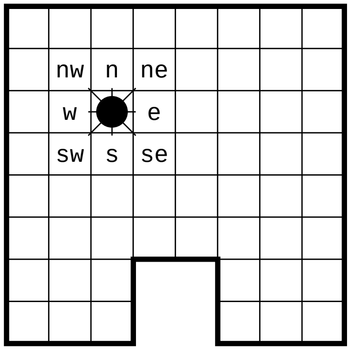
- Genetic algorithms only allow us to evolve some part of the agent program.
- We need to code up the "chromosome" and decode to get the agent itself.
- However, we can do evolution on more complex objects.
- In genetic programming we do evolution on programs themselves.

- We can't get completely away from some representation:



- However, in a suitable language (Lisp) we can execute such functions directly.

- Other languages will need a little translation.
- Let's look at how GP can be used to evolve the wall following robot.



- We build the program up from four primitive functions:
  1. `AND(x, y) = 0 if x = 0; else y`
  2. `OR(x, y) = 1 if x = 1; else y`
  3. `NOT(x) = 0 if x = 1; else 1`
  4. `IF(x, y, z) = y if x = 1; else z`

  and four actions:
  1. north move one cell up the grid
  2. east move one cell right in the grid
  3. south move one cell down the grid
  4. west move one cell left the grid

Note

We must ensure that all expressions and sub-expressions have values for all possible arguments, or terminate the program.

This ensures that any tree constructed so a function is correctly formed will be an executable program.

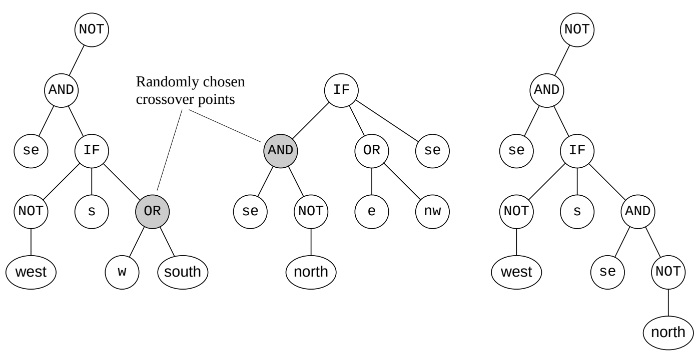
Even if the program is executable, it may not produce "sensible" output.

It may divide by zero, or generate a negative number where only a positive number makes sense (as when setting a price).

So we always need to have some kind of error handling to deal with the output of individual programs..

Reproduction

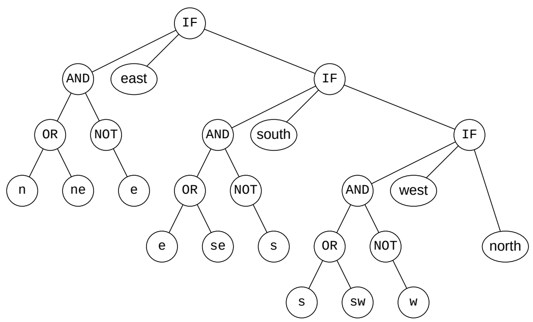
- The basic way we do GP is like GA.
- We have a fitness function,
- Do selection of the most fit,
- Breed them.
- But how do we breed programs?



Mother program Father program Child program

Before we start

- To give us an idea of what we are looking for, the following slide gives an example program in the GP tree-format.
- This program (check it) implements the same wall following program that we looked at in the "stimulus response" lecture.
- This shows that the GP-format is somewhat clumsy.
- However, as we shall see, this program is relatively compact when compared with the programs that will be generated by GP.




```
(IF (AND (OR (n) (ne)) (NOT (e)))
    (east)
    (IF (AND (OR (e) (se)) (NOT (s)))
        (south)
        (IF (AND (OR (s) (sw)) (NOT (w)))
        (west)
        (north))))
```

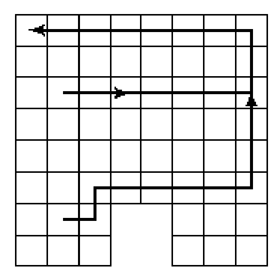

To evolve, perchance to follow walls

- We start with 5000 random programs.
- Fitness is evaluated by running on the task.
- Run the program 60 times and count the number of cells next to the wall visited.
- Worst possible program gets 0.
- Best possible program gets 32.
- Do 10 runs from random start points.
- Total count is the fitness.

- Then we need to breed.
- Take 500 programs and add them to the next generation.
- Choose them by *tournament selection*:
  - pick 7 at random;
  - add the most fit to the next generation.
- Then create 4500 children into the next generation—parents chosen by tournament selection.
- Mutate (?) by replacing a randomly chosen subtree with a random subtree.

Generation 0

The most fit member of the randomly generated initial programs has a fitness of 92, and has the kind of behaviour shown below.

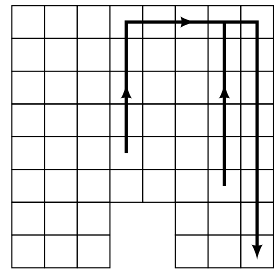


The program itself is given in the next slide.

```
(AND (NOT (NOT (IF (IF (NOT (nw))
                       (IF (e)(north) (east))
                       (IF (west)(0) (south))
                   (OR (IF (nw)(ne)(w))
                       (NOT (sw)))
                   (NOT (NOT (north)))))
     (IF (OR (NOT (AND (IF (sw)(north)(ne))
                       (AND (south)(1))))
             (OR (OR (NOT (s))
                     (OR (e)(e)))
                 (AND (IF (west)(ne)(se))
                      (IF (1) (e)(e)))))
         (OR (NOT (AND (NOT (ne))(IF (east)(s)(n))))
             (OR (NOT (IF (nw)(east)(s)))
                 (AND (IF (w)(sw)(1))
                      (OR (sw)(nw)))))
         (OR (NOT (IF (OR (n)(w))
                      (OR (0)(se))
                      (OR (1)(east))))
             (OR (AND (OR (1)(ne))
                      (AND (NW)(east)))
                 (IF (NOT (west))
                     (AND (west)(est))
                     (IF (1)(north)(w))))))))
```


Generation 2

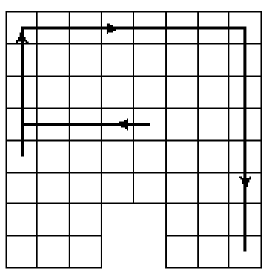
The most fit member of generation 2 has fitness 117.



```
(NOT (AND (IF (ne)
              (IF (se)(south)(east))
              (north))
          (NOT (NOT (e)))))
```


Generation 6

The most fit member of generation 6 has fitness 163.



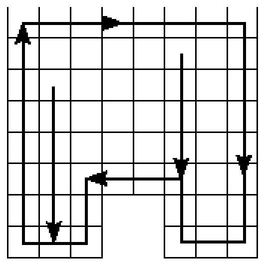
The program follows the wall perfectly, but gets stuck in the bottom righthand corner.

the best program from generation 6.


```
(AND (NOT (NOT (IF (IF (NOT (nw))
                       (IF (e)(north) (east))
                       (IF (west)(0) (south))
                   (OR (IF (nw)(ne)(w))
                       (NOT (sw)))
                   (NOT (NOT (north)))))
     (IF (OR (NOT (AND (IF (sw)(north)(ne))
                       (AND (south)(1))))
             (OR (OR (NOT (s))
                     (OR (e)(e)))
                 (AND (IF (west)(ne)(se))
                      (IF (1) (e)(e)))))
         (OR (NOT (AND (NOT (ne))(IF (east)(s)(n))))
             (OR (NOT (IF (nw)(east)(s)))
                 (AND (IF (w)(sw)(1))
                      (OR (sw)(nw)))))
         (OR (NOT (IF (OR (n)(w))
                      (OR (0)(se))
                      (OR (1)(east))))
             (OR (AND (OR (1)(ne))
                      (AND (NW)(east)))
                 (IF (NOT (west))
                     (AND (west)(est))
                     (IF (1)(north)(w))))))))
```

Generation 10

The most fit member of generation 10 has fitness of close to the maximum 320.

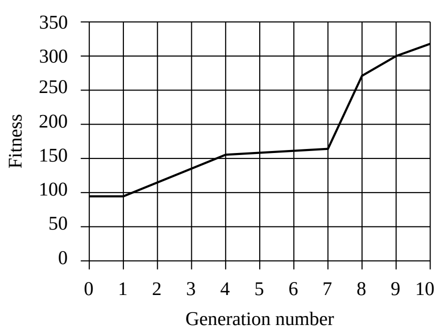


The program follows the wall perfectly, heading south until it reaches the boundary.

the best program from generation 10.

```
(IF (IF (IF (se)(0)(ne))
        (OR (se)(east))
        (IF (OR (AND (e)(0))
                (sw))
            (OR (sw)(0))
            (AND (NOT (NOT (AND (s)(se))))
                 (se))))
    (IF (w)
        (OR (north)
            (NOT (NOT (s))))
        (west))
    (NOT (NOT (NOT (AND (IF (NOT (south))
                            (se)
                            (w))
                        (NOT (n)))))))
```


This graph shows how the fitness of individuals grows quite sharply over the ten generations.



Summary

- This lecture has introduced *evolutionay computing* techniques.
- These are techniques in which (parts of) agents evolve.
- We looked at two techniques:
  - Genetic algorithms
  - Genetic programming
- Note that evolutionary techniques are sometimes taken to include neural networks.
- Genetic algorithms and genetic programming give us a way to learn in an *unsupervised way*.