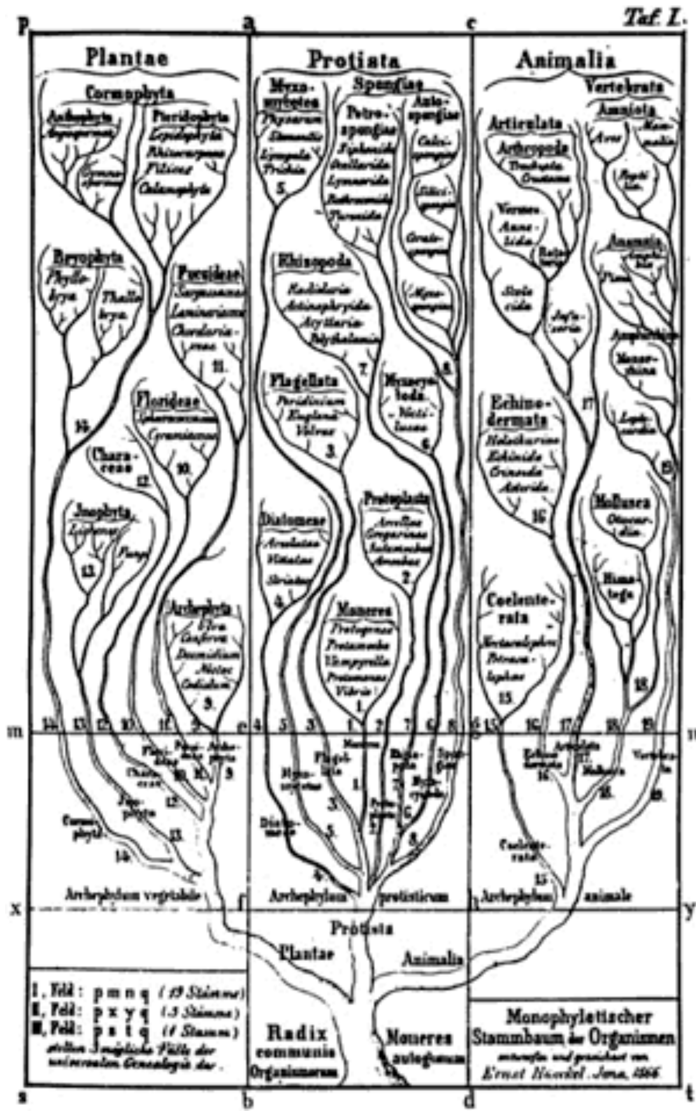


Artificial Evolution

FIT3094 AI, A-Life and Virtual Environments
Alan Dorin

MONASH University
Information Technology

Copyrighted imagery used in the preparation of these lecture notes remains the property of the credited owners and is included here for educational purposes only. It has been sourced where possible from publicity material or from material placed within the public domain.

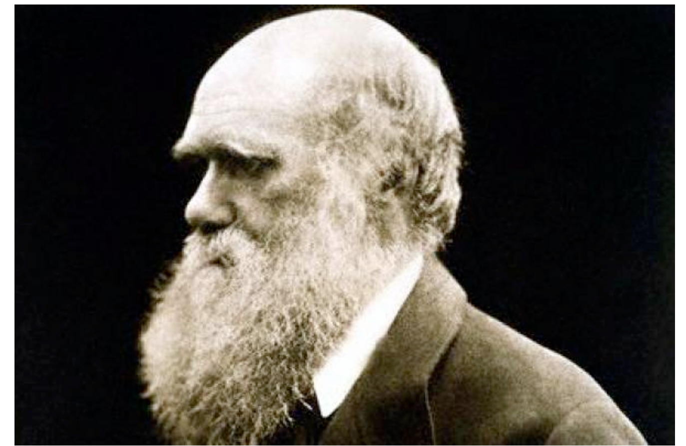
Learning Objectives

To understand the basis of evolutionary computation.

To understand the evolutionary algorithm and the kinds of problems it may be applied to.

To be able to explain the processes of crossover and mutation.

Evolution by Natural Selection



Charles Darwin, 1809 – 1882

In *On the Origin of Species by Means of Natural Selection* (1859),
Charles Darwin* proposed his theory of evolution.

Offspring inherit traits from their parents.
Variations in organisms occur in reproduction.
Variations may make an organism more or less likely to produce offspring.

Variations that allow organisms to successfully reproduce will be preserved in successive generations.

Variations that do not, will not be preserved, since the offspring carrying them will be less likely to produce offspring themselves than the organisms carrying helpful variations.

* Alfred Russel Wallace had a similar theory at the same time

Artificial Evolution

We can extract the processes of

inheritance,
reproduction and
selection

from nature, and implement them in software to arrive at one of several methods of *evolutionary computation* (also known as *artificial evolution*).

These methods make excellent search and optimisation algorithms for many difficult game-related and virtual environment design problems.

Finding paths through maps.
Finding arrangements of game pieces that meet particular requirements.
Designing game agent shapes and visual characteristics.
Designing complex agent behavioural strategies.

Could you write Finite State Machines to...

...have these creatures detect and run from predators, flock together, search for food and eat it, stand-up without falling over etc.?

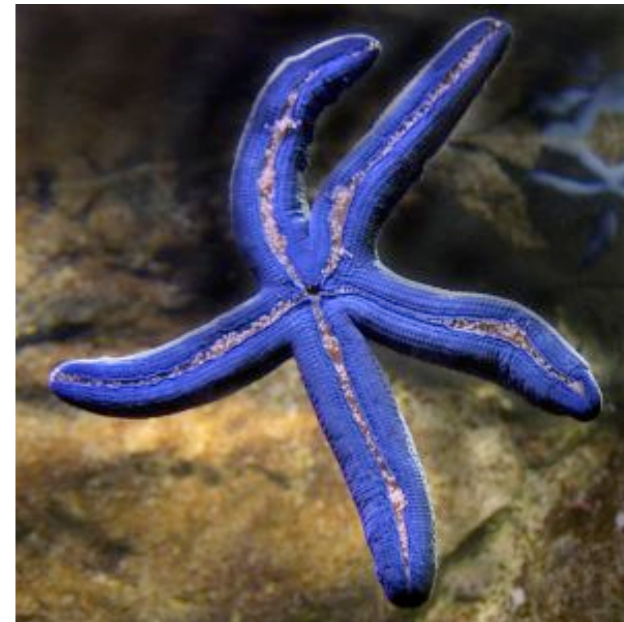
...have this creature detect and chase the weakest prey, catch and eat it, stand-up without falling over, avoid being eaten by predators larger than itself etc.?

Evolutionary Computation could be used to do all of this. It is a *searching* technique good at finding solutions to complex problems where incremental improvements to solutions exist.

Images by Ben Porter, 2009

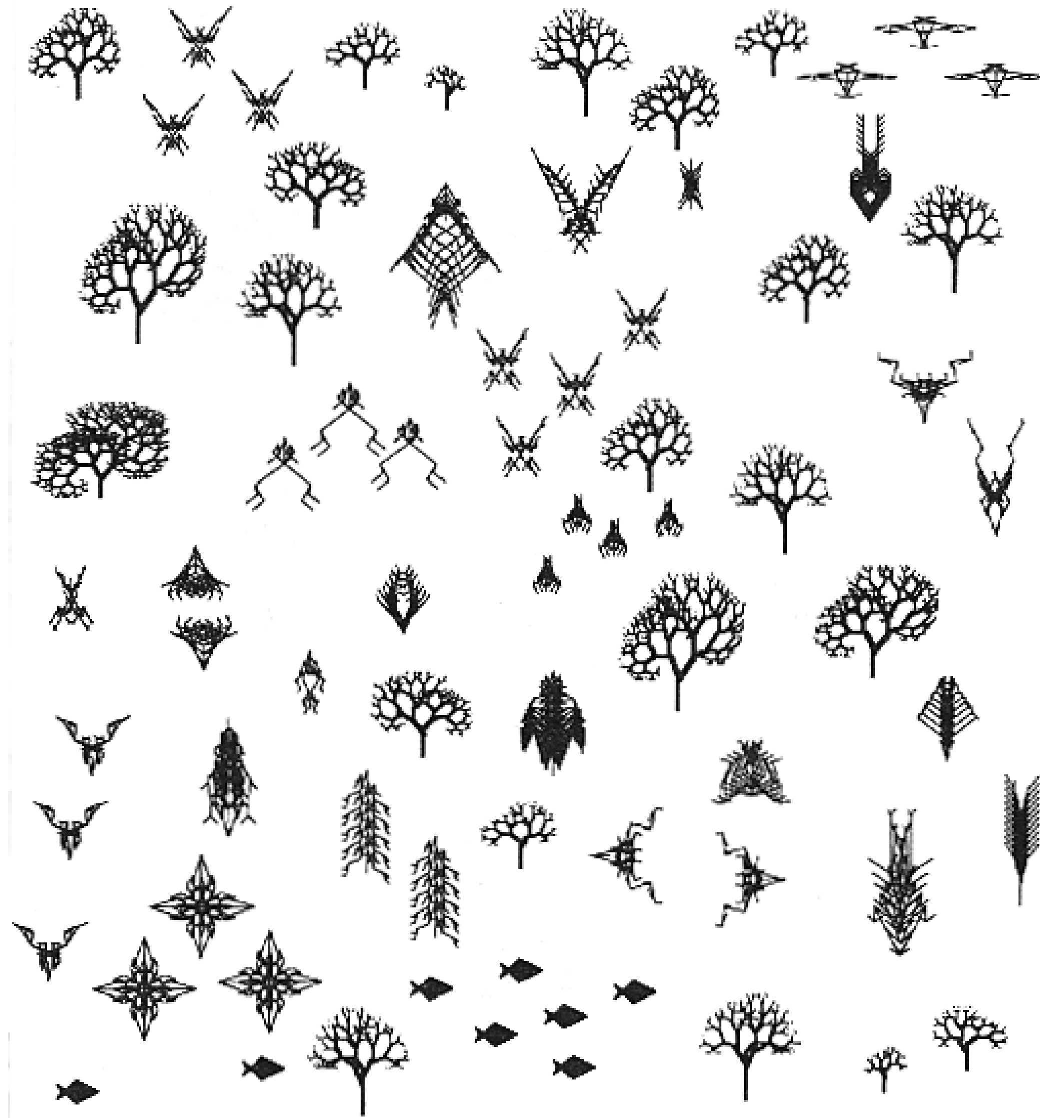
Could you design a 5 legged creature and a method to coordinate its gait*
that does not tie its legs in knots and end with the creature tripping over
its own feet?

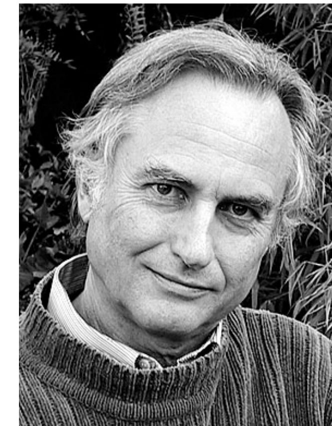
Evolutionary Computation could be used to do this.

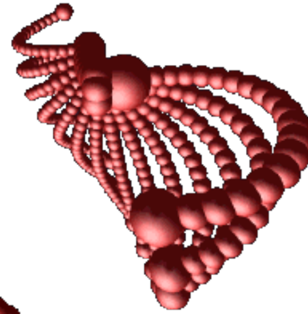


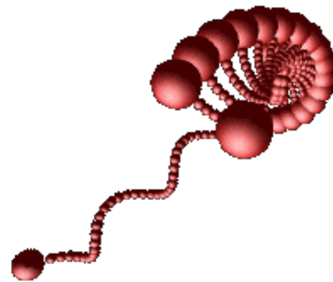
* *gait*, a manner of walking

Remember Richard Dawkins? Here are some structures he generated using artificial evolution.

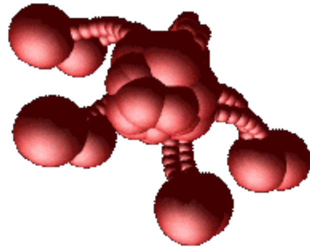






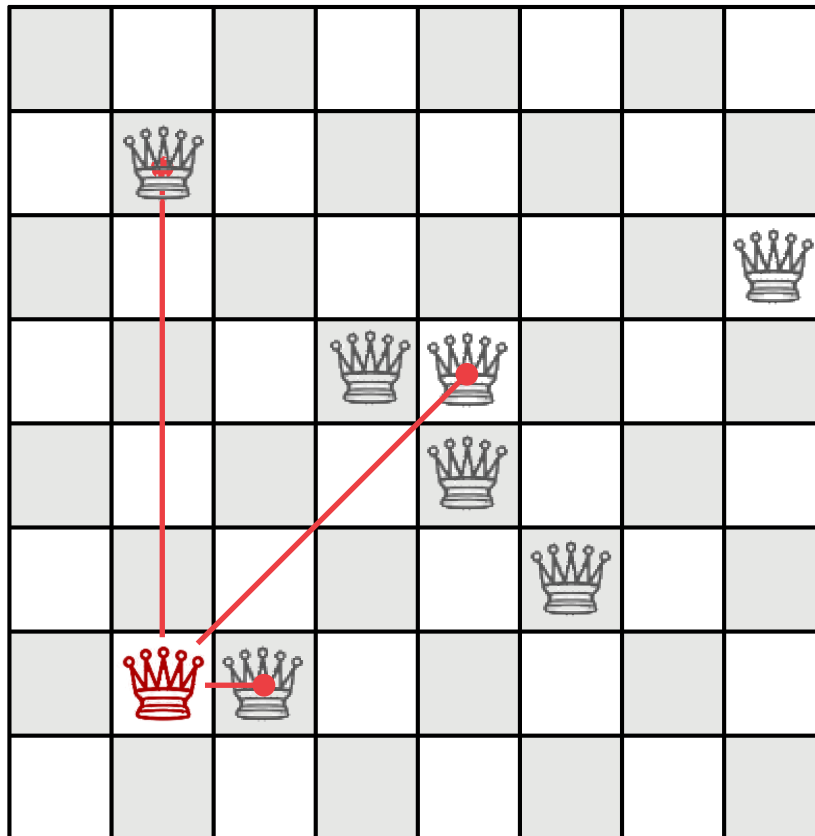


Here are some 3D structures generated using artificial evolution.



Artificial Evolution, the *8 Queens* problem

Can you find an arrangement of 8 queens on the chess board in such a way that no queen is attacking any other?



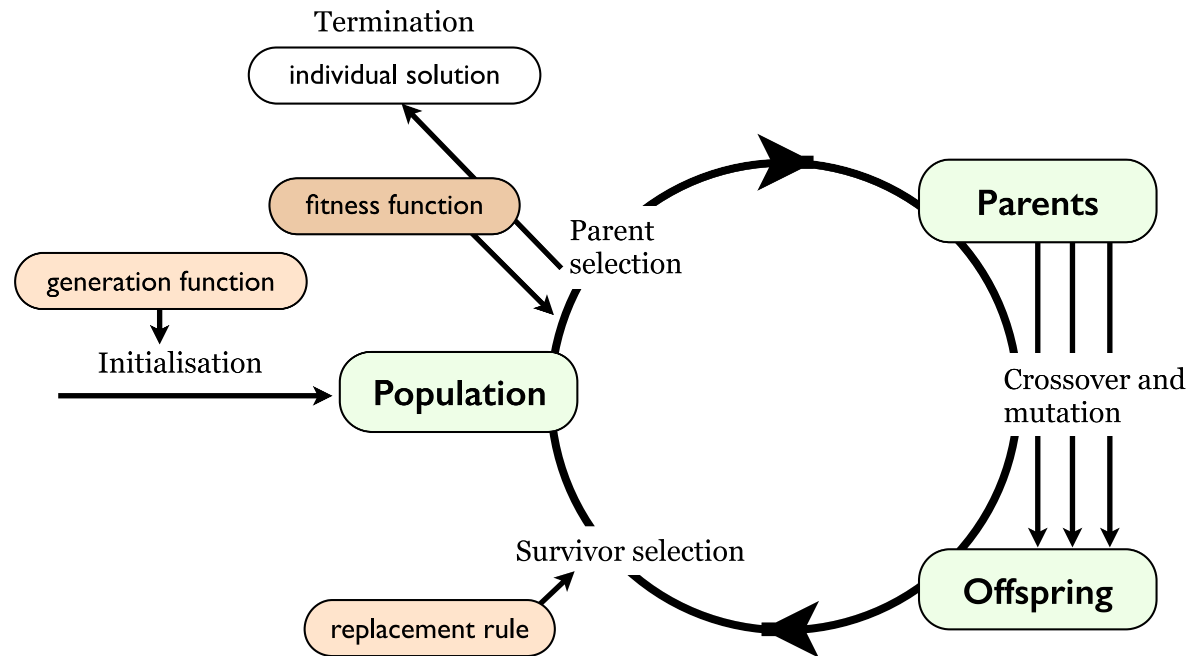
Queens can attack along complete rows, columns and diagonals.

For instance, the red queen is attacking three others, as shown.

We shall look at how Evolutionary Computation can be used to do this.

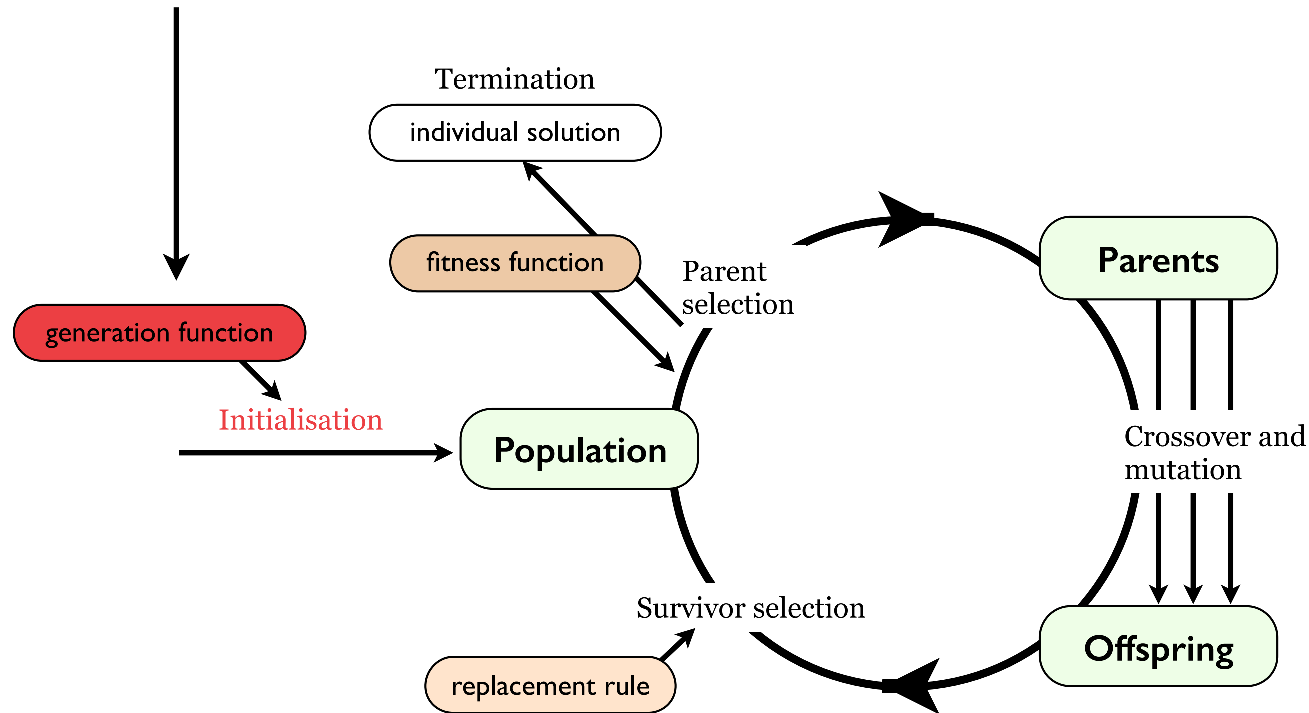
Artificial Evolution, the algorithm

Artificial Evolution follows this general algorithm.



The generation function

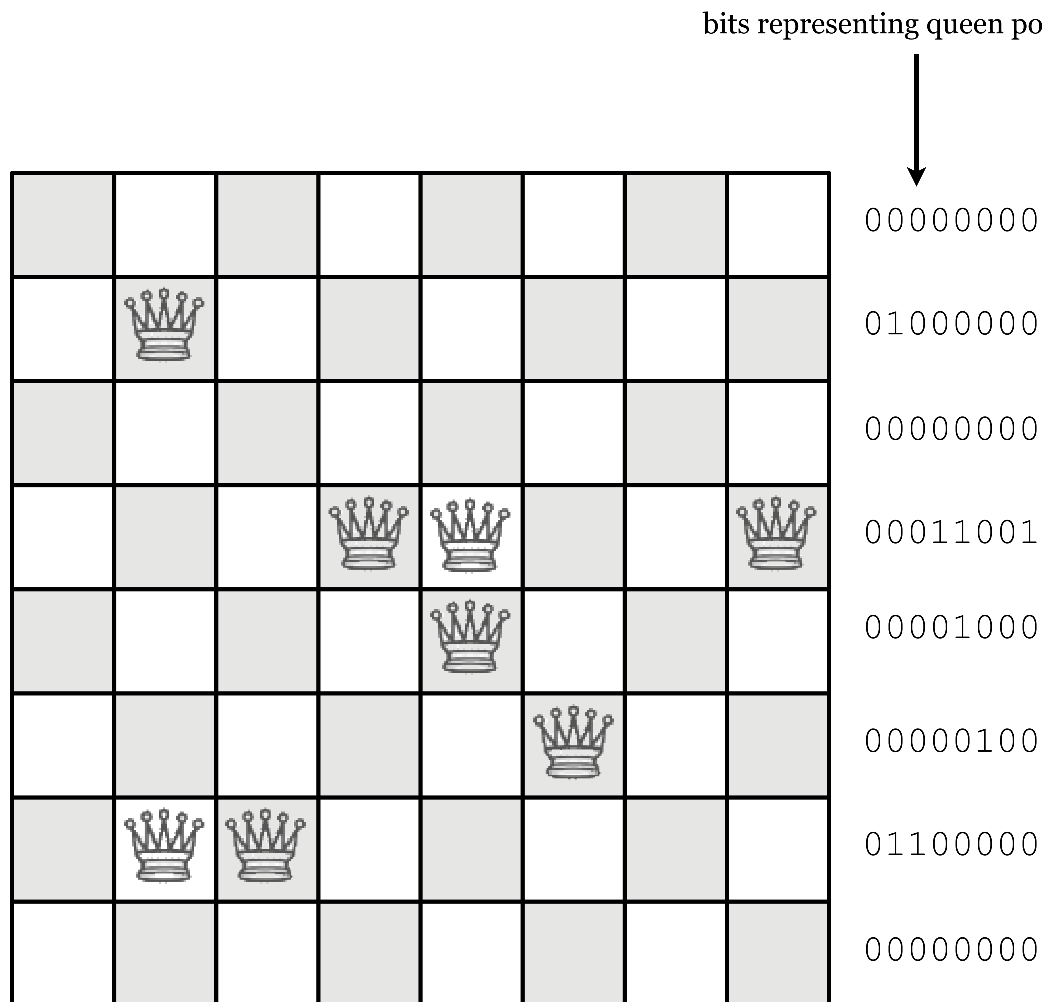
The algorithm is initialised here, by randomly generating perhaps 100 potential problem solutions.



For instance, the initial population in the 8 Queens problem might be a set of random arrangements of queens on the board.

Artificial Evolution, the *8 Queens* problem

How can we represent these potential solutions to the 8 Queens problem in a data-structure?



These bits can be strung together into a single 64-bit number that can encode any configuration.

This is called the *genotype* or just the *genes* of the solution.

The board itself, with the queens placed on it, some attacking one another, some not, is called the *phenotype*.

Artificial Evolution, the *8 Queens* problem

But, for the 8-queens problem, only numbers with eight 1s and fifty-six 0s are valid board configurations.

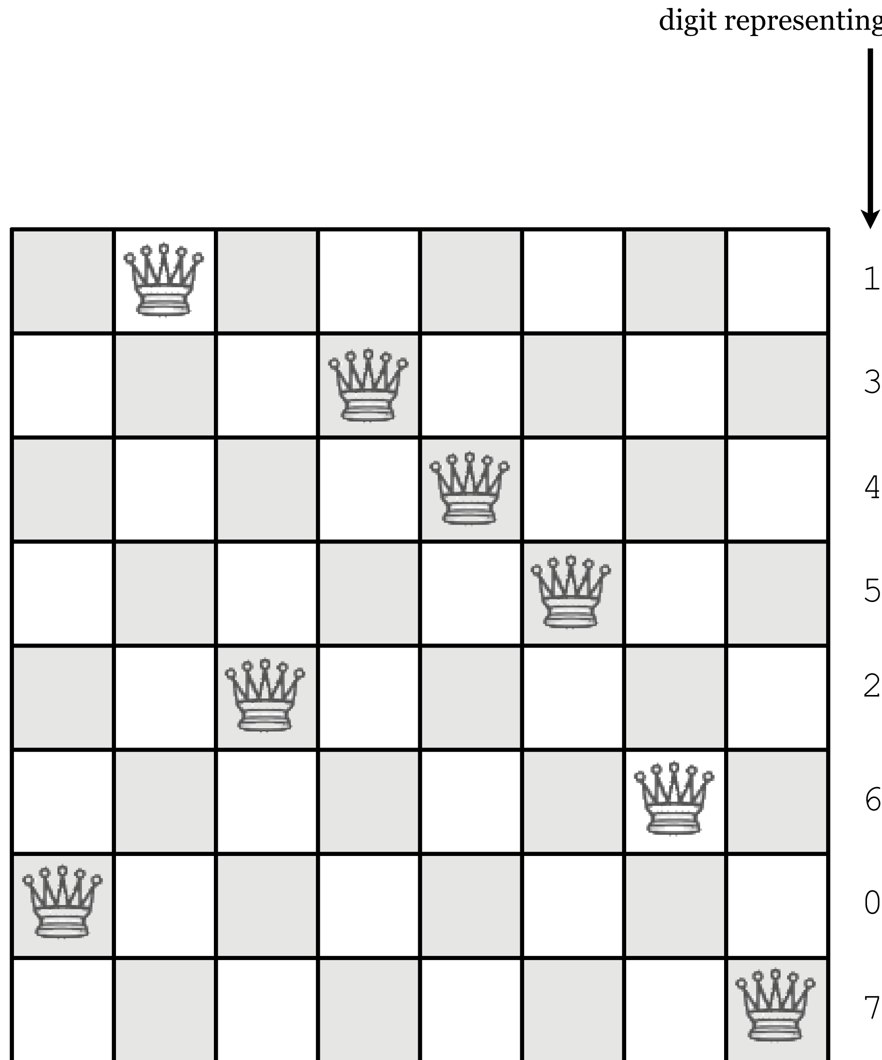
Also, we know that a valid solution will never have two queens in the same row (or the same column for that matter).

So, what is a better way to represent the boards that ensures only potentially *useful* board configurations can be represented?

Why would we want such a representation for solving this puzzle?

Artificial Evolution, the *8 Queens* problem

How can we represent potential solutions to the 8 Queens problem in a data-structure?



This board configuration can be represented by a single 8-digit number where all digits are between 0 and 7: `13452607`.

This is a different way to encode a genotype for the 8 Queens problem.

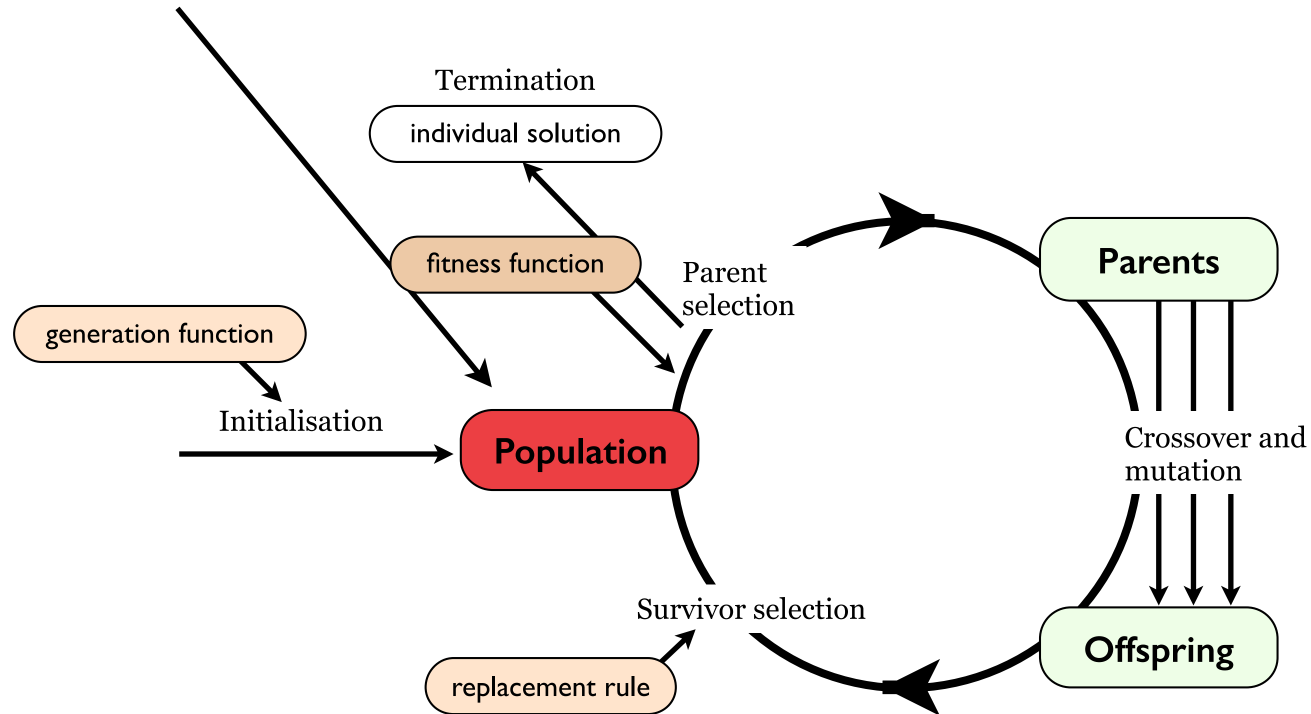
Note that in the different board configuration shown here, no queens share a row or column... but this is still *not* a complete solution as queens can attack along diagonals.

Still, it is a *better* configuration than the previous one, since less queens are attacking one another.

Evolutionary computing works on improving solutions incrementally. It depends on being able to compare solutions.

The initial population

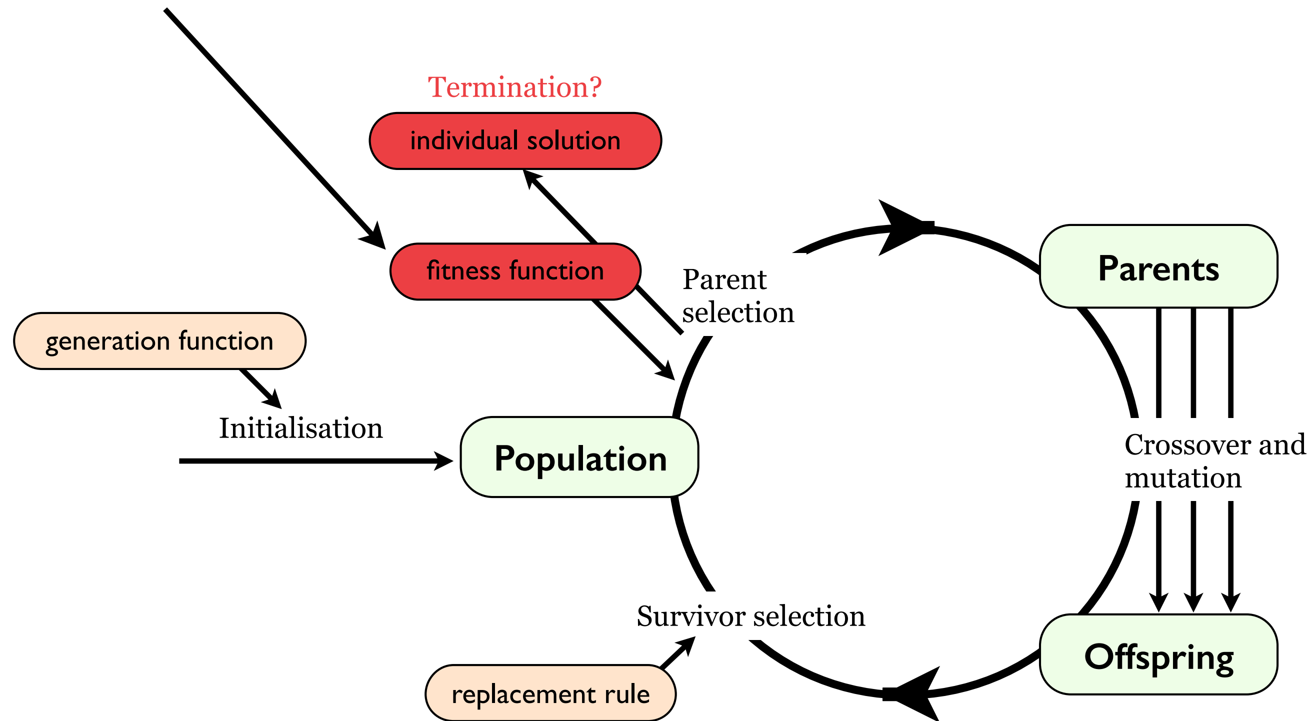
These randomly generated potential solutions, each encoded as a data-structure, form the initial "population" of solutions to the problem.



The fitness function

The population of potential solutions is each evaluated using a *fitness function*. This returns a score for the potential solution based on how well it satisfies the goal. The better the solution, the higher the fitness score it receives.

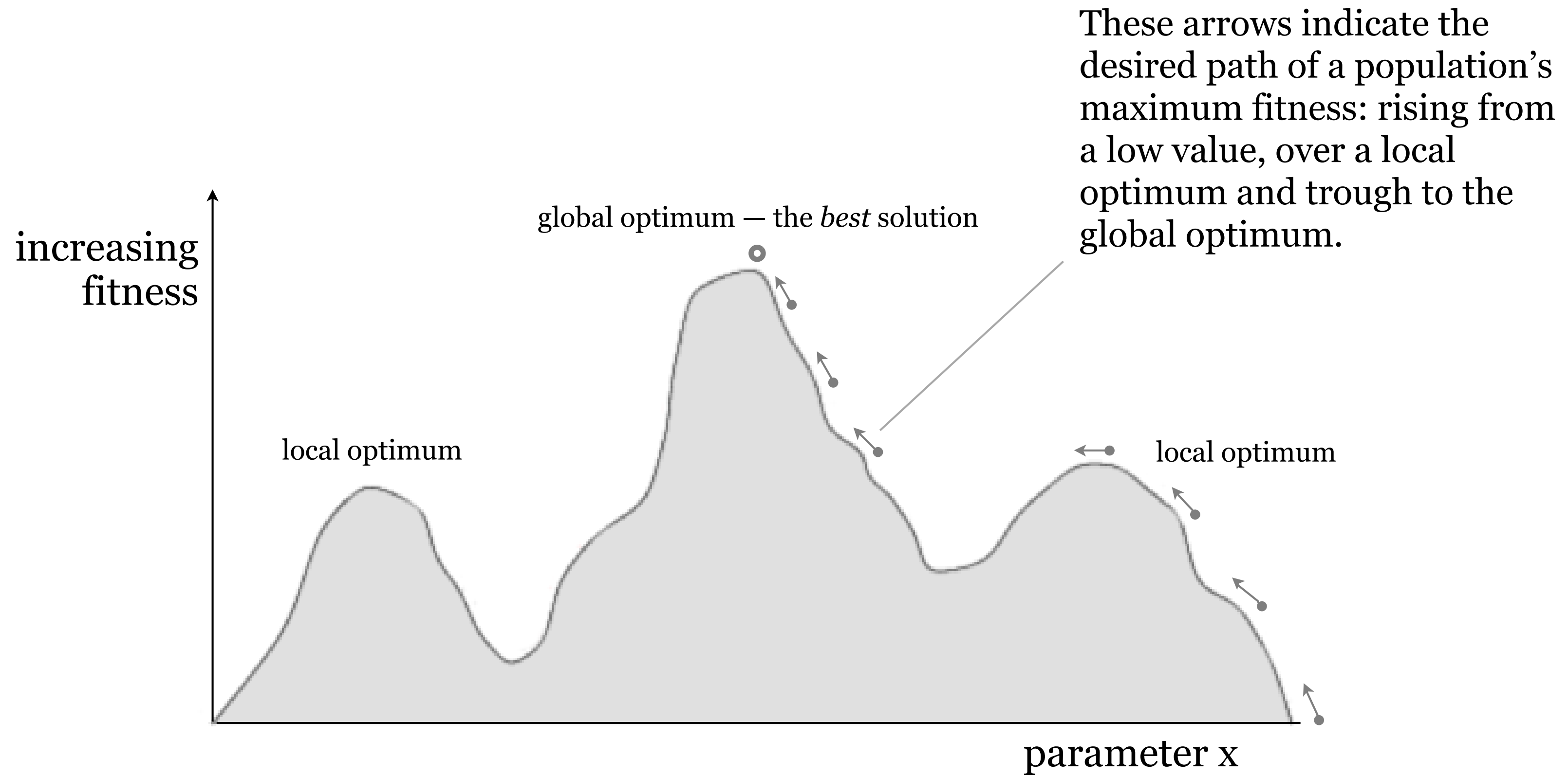
For instance, the fitness measure in the 8 Queens problem might be a the number of *non-attacking* pairs of queens. This has a value of 28 for a solution.



If a member of the population receives a perfect fitness score (or the algorithm has looped enough times so that the programmer thinks it is time to finish), the algorithm can terminate!

It has found a perfect solution (or it has found the best solution it is ever going to find).

The fitness landscape

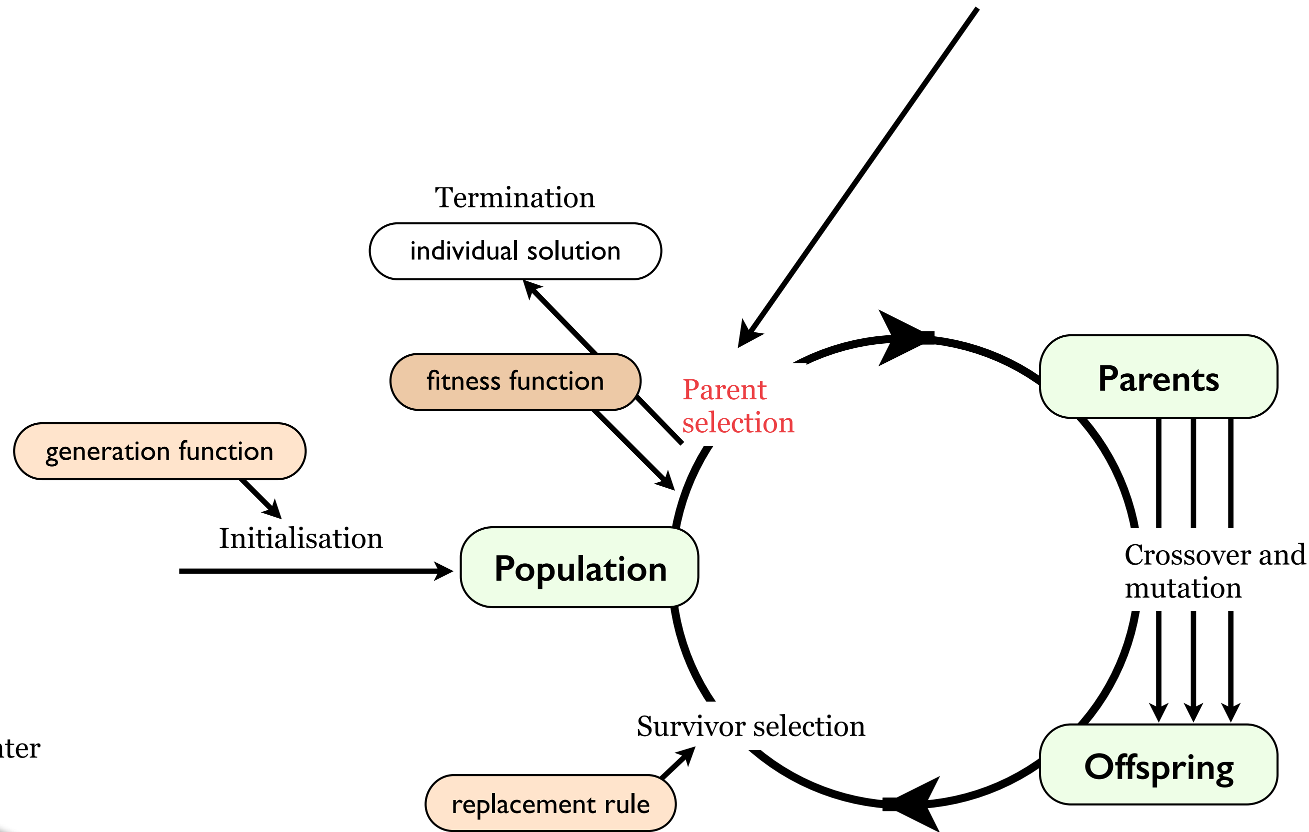


Here is a sketch of a (imaginary) fitness landscape. The values of parameter x as tested by the genetic algorithm give rise to troughs and peaks of fitness giving the appearance of a landscape.

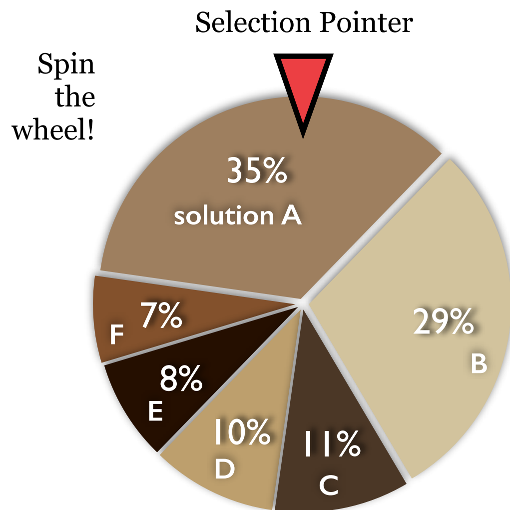
Parent selection

The higher the fitness score a potential solution receives, the greater the *probability* that it will be selected to become a parent.

I.e. The fittest individual in the population would be expected to parent many more children than the least fit member of the population.



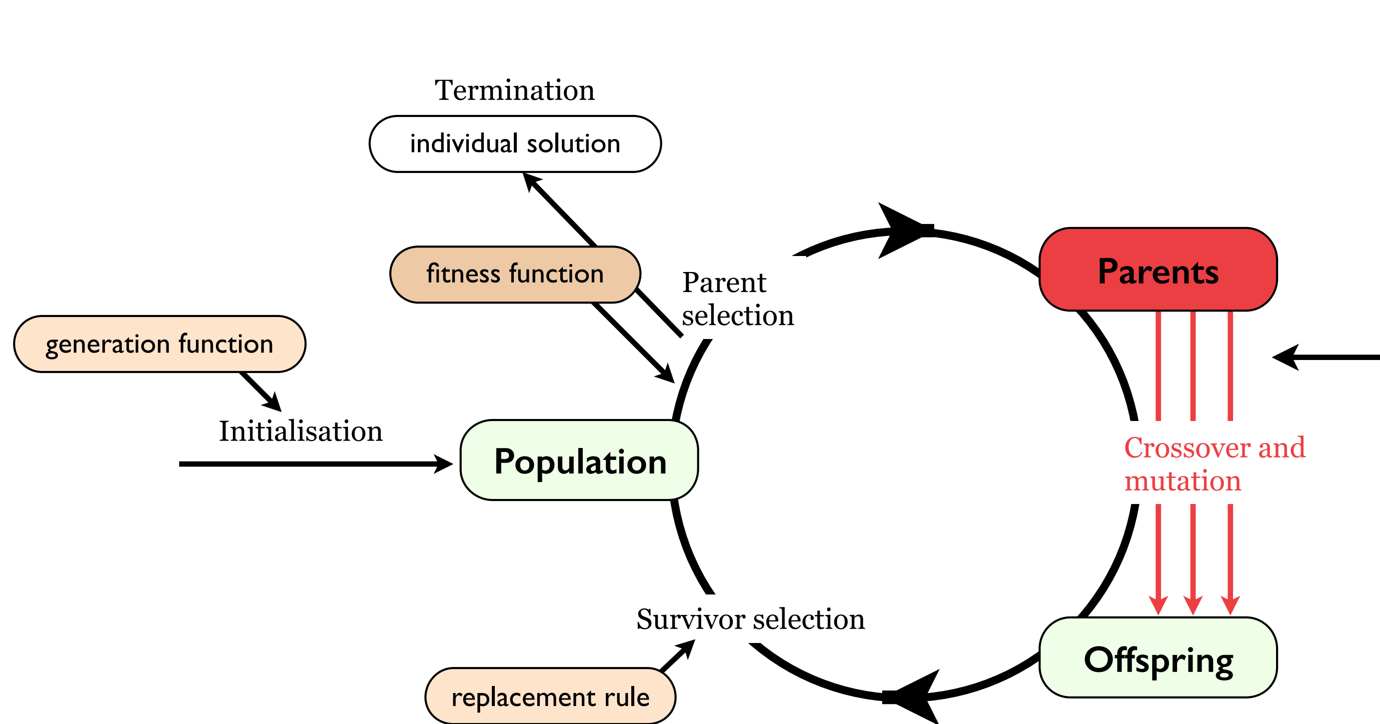
Spin the wheel!



Different methods for using the fitnesses of solutions to select parents exist. One simple method is called *roulette wheel selection* using a *fitness-proportionate* or a *ranked* system to select the wedge sizes.

Reproduction

Pairs of selected parents "reproduce" — they mate and give birth to a child. Reproduction involves splicing together the characteristics of the two parents (crossover) and sometimes, a random change in one of the characteristics of the child (mutation).

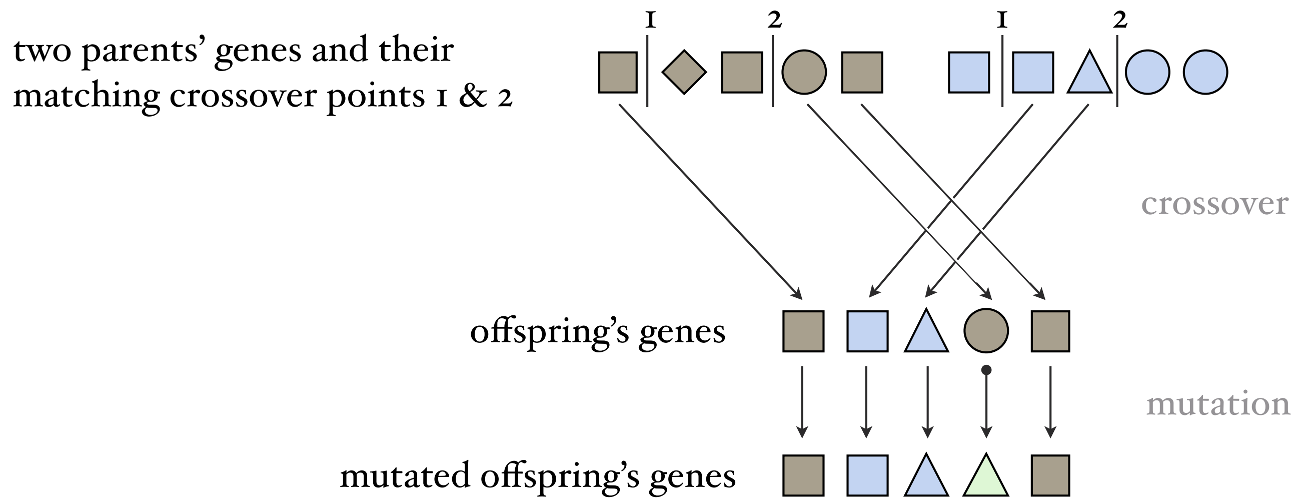


Crossover and Mutation

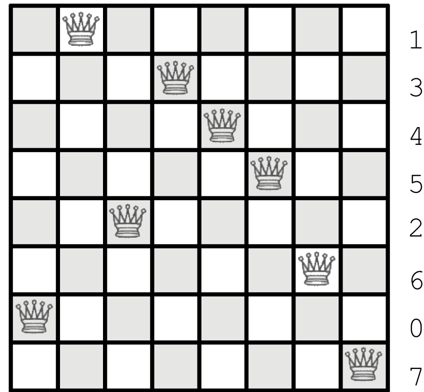
During reproduction, two parent solutions are mated together.
The operations that occur to produce a child are *crossover* (recombination) and *mutation*.

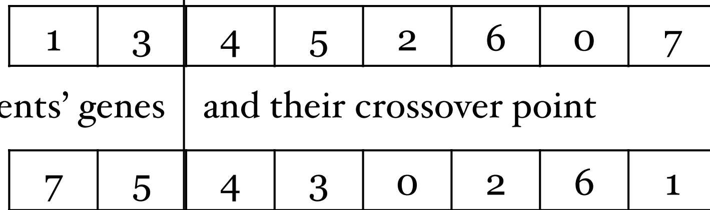
Crossover requires one or more locations within the genotype to be selected randomly. The location of these *crossover points* determines which genes from each parent are spliced together to appear in their offspring.

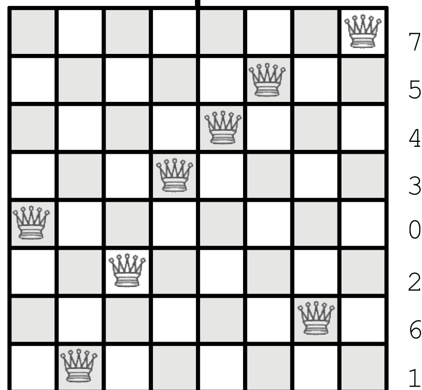
Mutation randomly varies one of the child's genes or discards it and generates a new one from scratch.

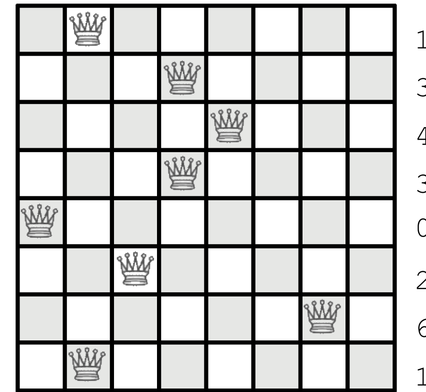


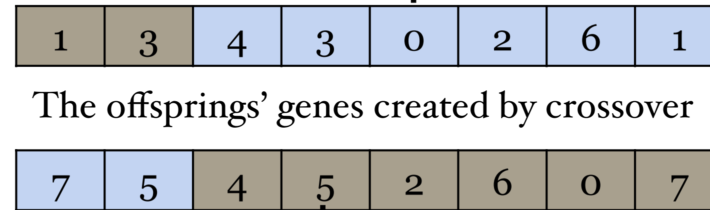
Crossover and Mutation

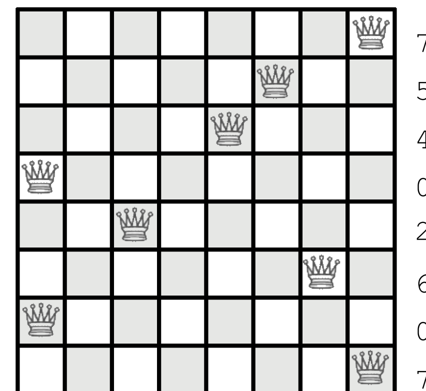




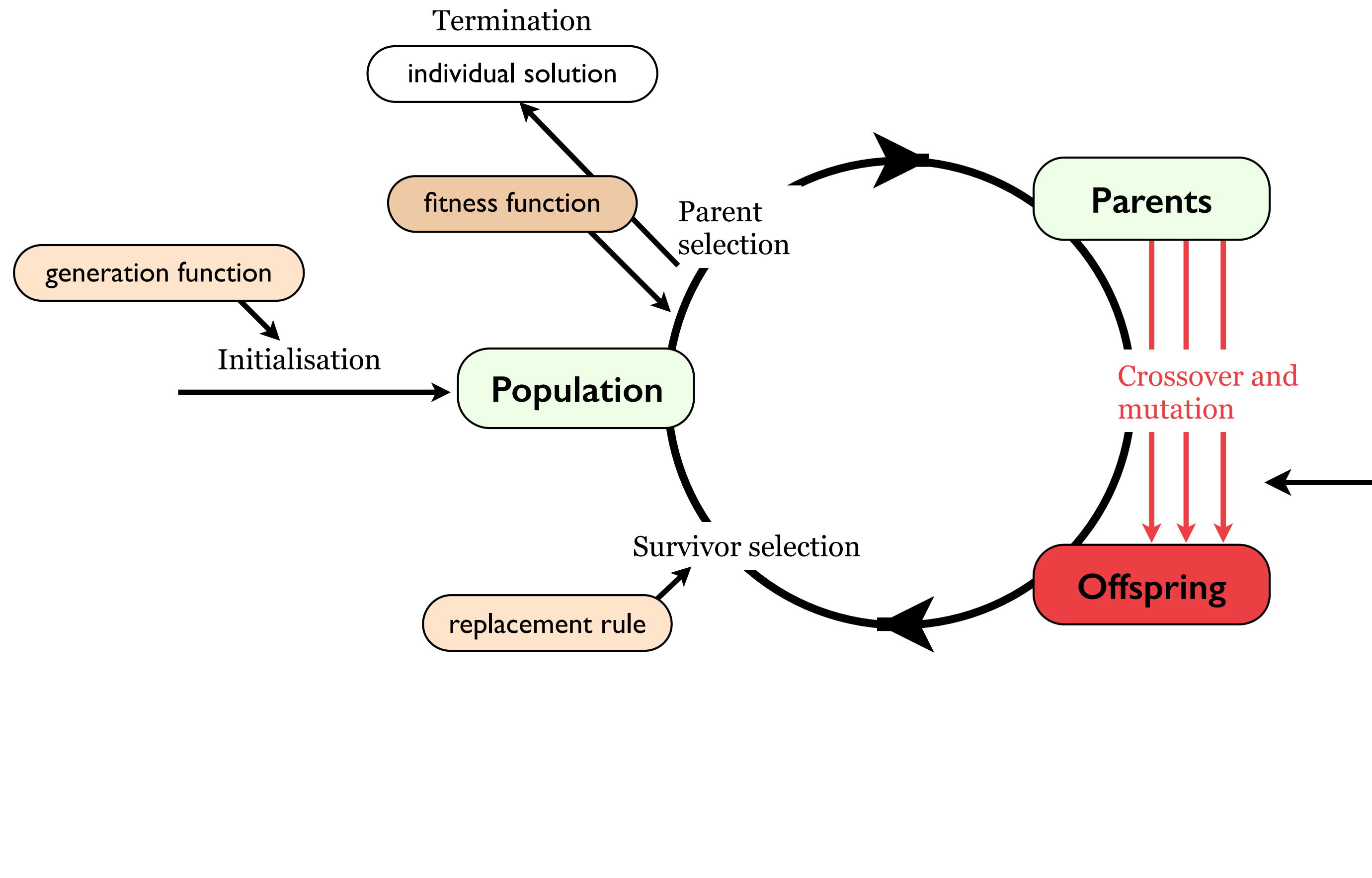






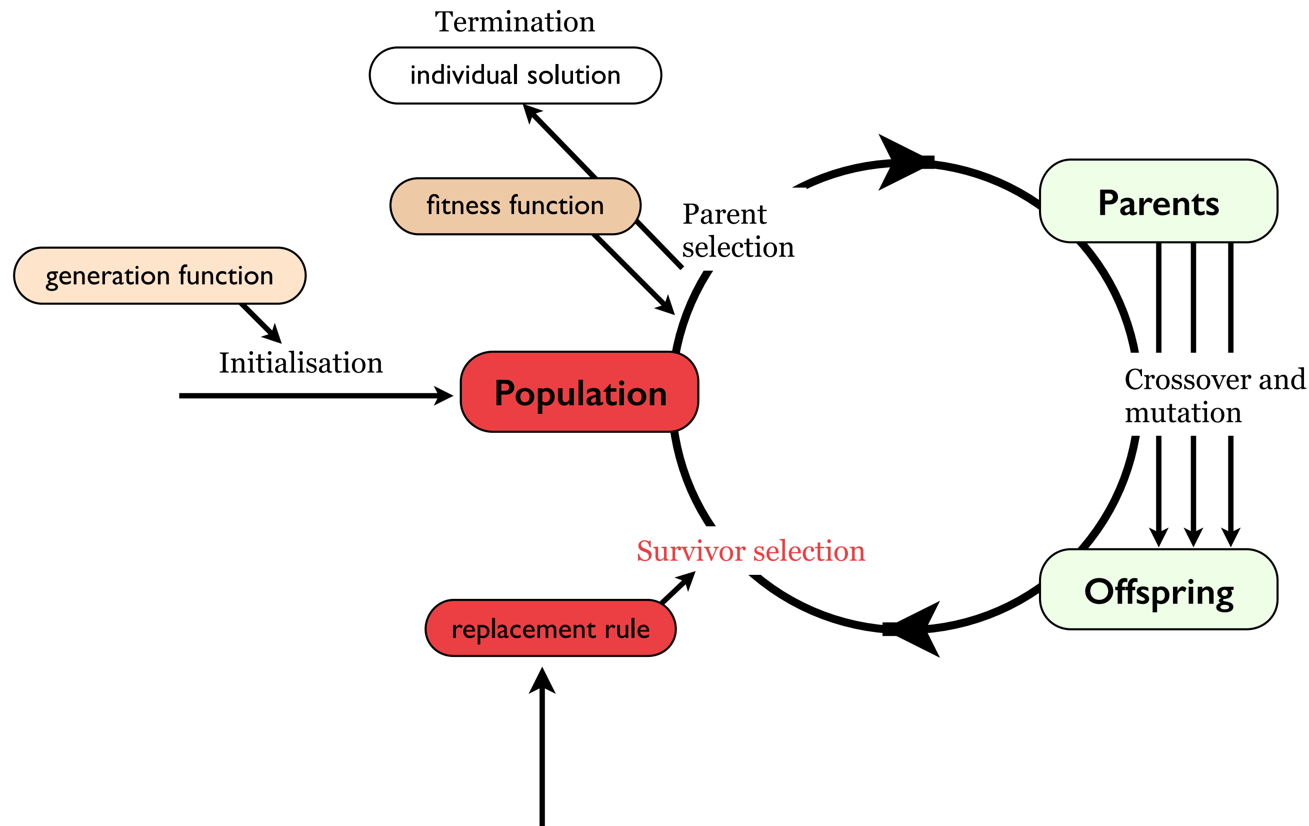


Offspring



Many pairs of parents reproduce children in this way
to produce a large number of offspring

Population replacement

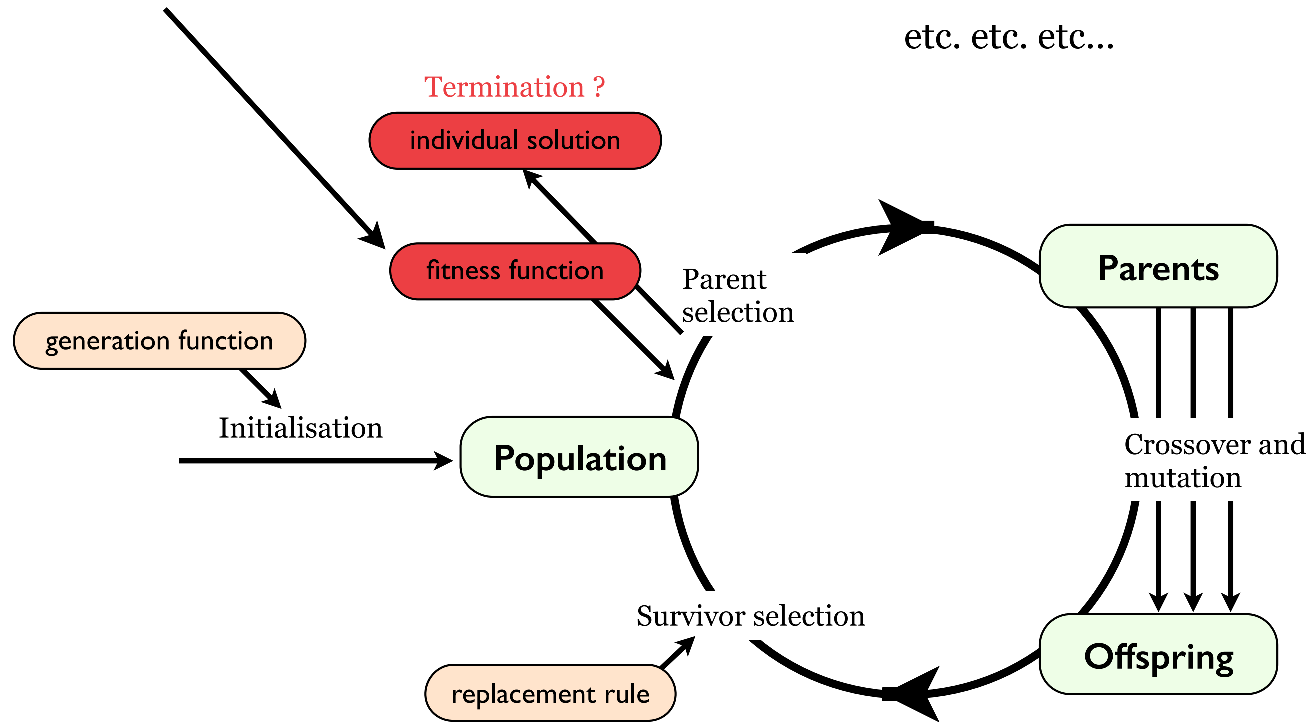


The algorithm requires that at each stage through the loop, the population size should remain constant. Hence, a *replacement rule* is used to decide which offspring should replace which members of the previous population.

Different replacement rules exist. E.g. Replace all of the population with new offspring, *or* select a random offspring and a random parent and whichever has the highest fitness value is kept.

Termination conditions

The population of potential solutions is each evaluated using a *fitness function*. This returns a score for the potential solution based on how well it satisfies the goal. The better the solution, the higher the fitness score it receives.

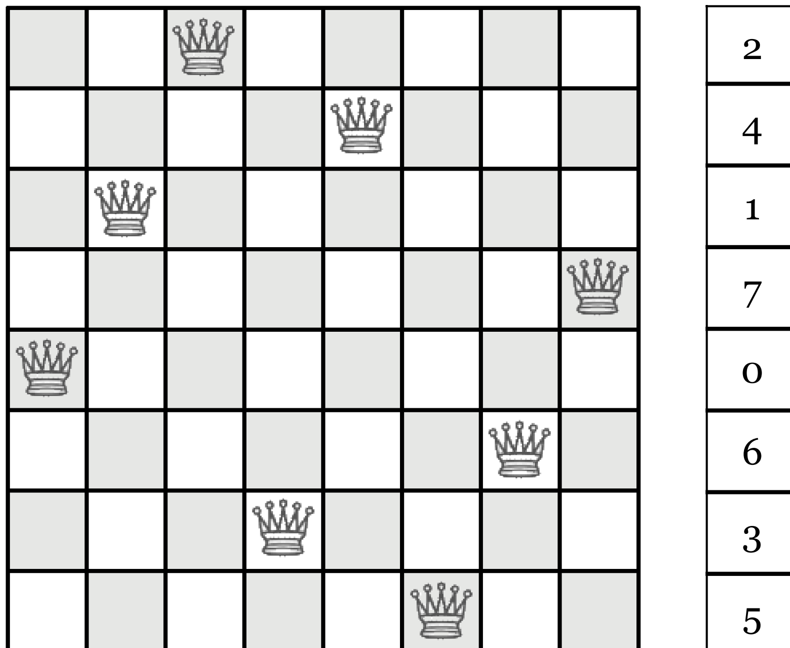


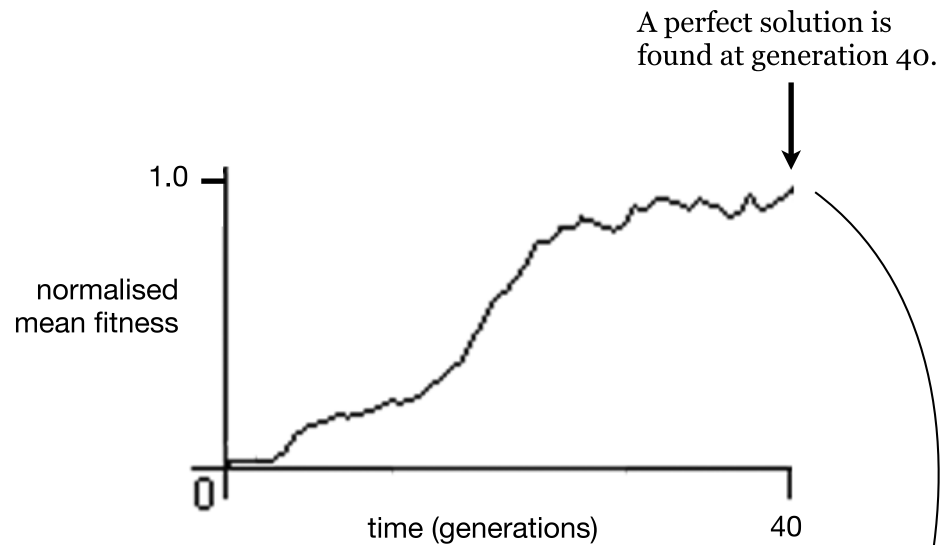
If a member of the population receives a perfect fitness score (or the algorithm has looped enough times so that the programmer thinks it is time to finish), the algorithm can terminate!

It has found a perfect solution (or it has found the best solution it is ever going to find).

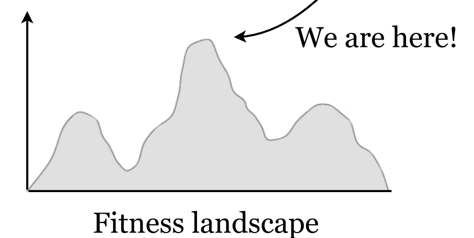
Fitness vs Time

As the algorithm repeatedly proceeds through the loop, generating and testing potential solutions, the average fitness of the population gradually improves.



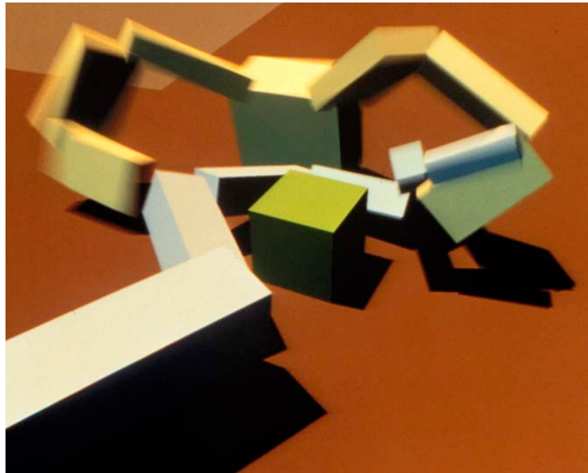


At some stage it is hoped that a perfect solution will be found. The algorithm can then stop.

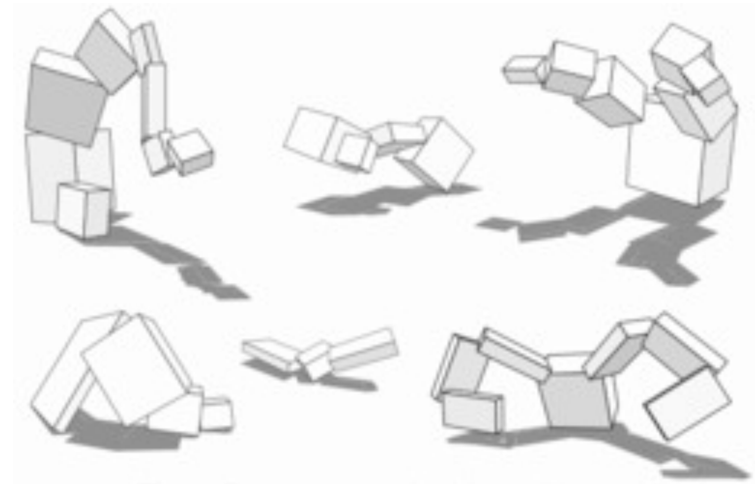


Fitness landscape

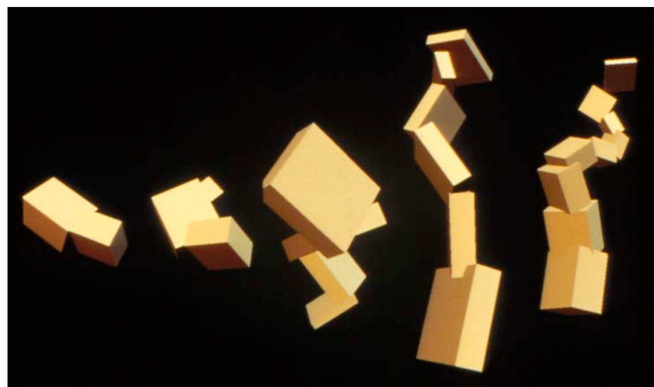
Here are some creatures with body structure and behaviour generated by an artificial evolution algorithm implemented by Karl Sims.



Creatures that compete to trap a puck.



Creatures that swim.

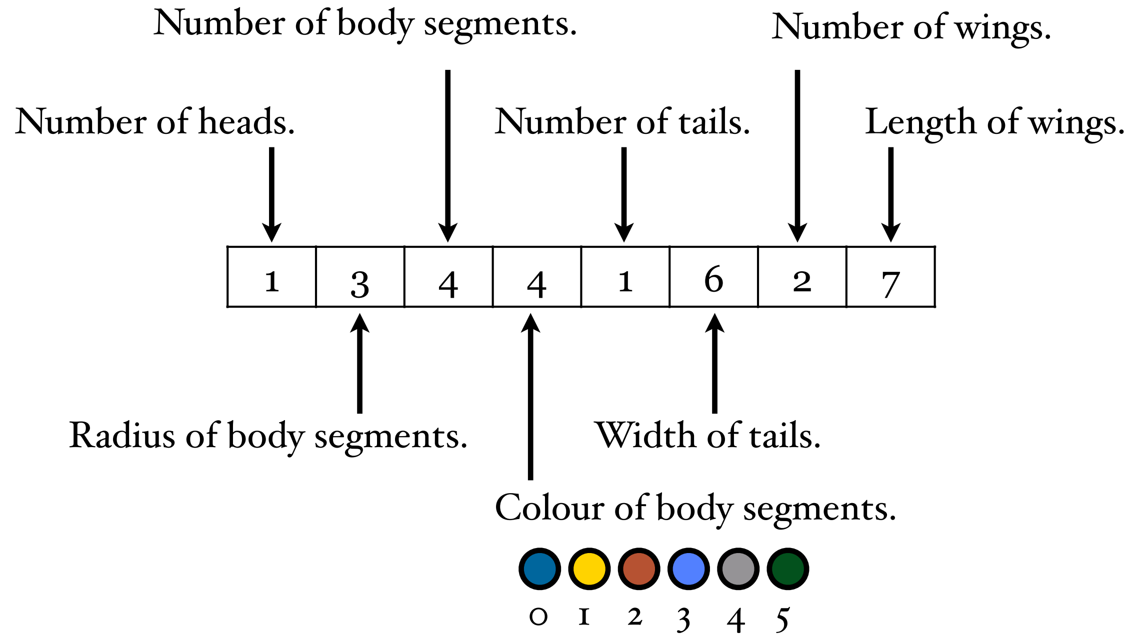


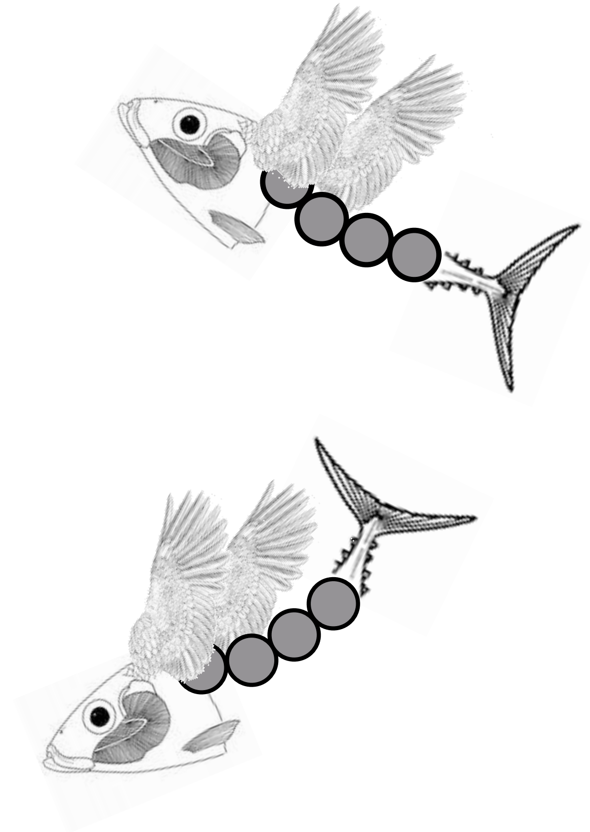
Creatures that walk.

What do you think he used as fitness functions?

Sims K., Evolving Virtual Creatures. *Computer Graphics*, Siggraph '94 Proceedings, July 1994, pp.15-22.

Adapting a simple genotype





Class discussion:

How could this basic genotype be used to specify a different structure of an agent or even its behaviour?

Alternative Applications specific to games.

Design complex agent controllers by evolving them to a simple level before release.
Users play the game against evolving controllers that adapt to their behaviour and changes in the game world.

However, evaluating the fitness of a population, breeding them and selecting members of the population for replacement is often CPU intensive.

This can be too much for a computer to do whilst running an interactive game so...

Design complex agent controllers by evolving them before release.
Then disable evolution algorithm.
Users play the game against the evolved controllers.

Have you met the learning objectives?

What is Evolutionary Computing and what biological process does it mimic?
Can you describe the evolutionary algorithm?

What are crossover and mutation? How do they work?

How might you use evolutionary computation to search for a finite state machine that chased a target on a grid world? How would you encode its genotype?

How could you specify the genotype of a creature like this?