Lecture 06 - Genetic Algorithms

Optimisation CT5141

James McDermott

NUI Galway

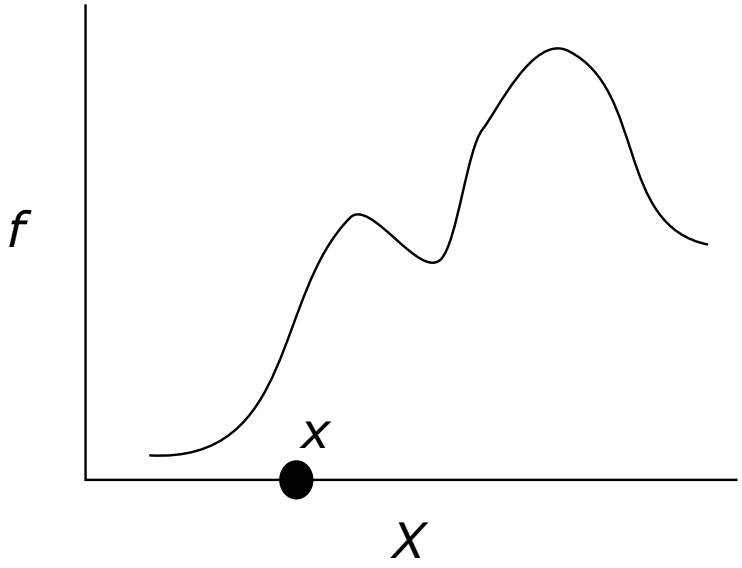
OÉ Gaillimh
NUI Galway

2020

Overview

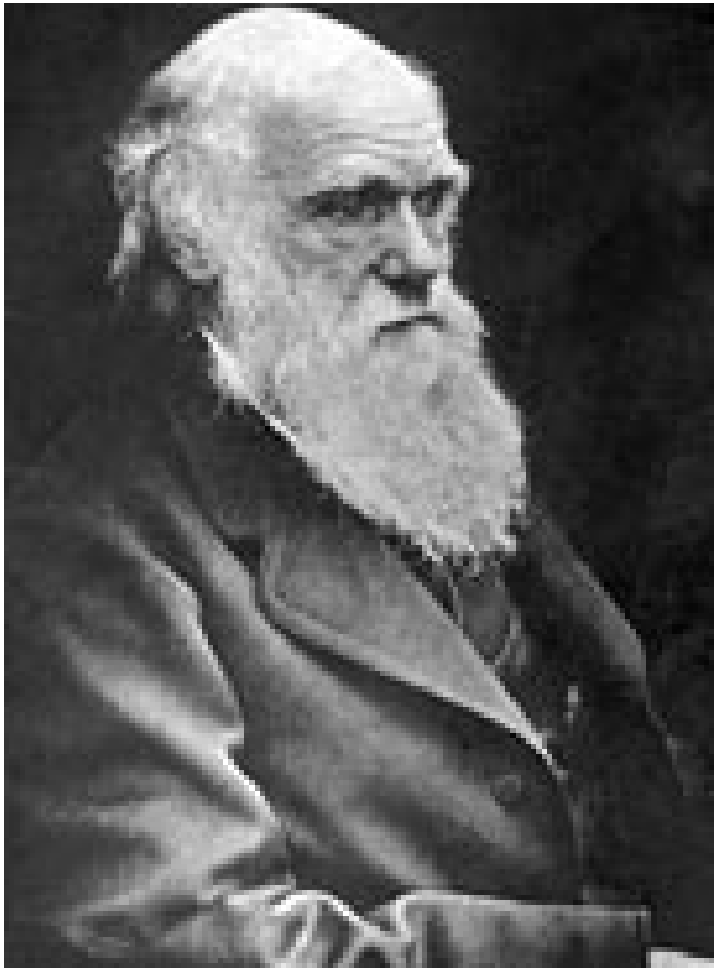
1. **Introduction to Genetic Algorithms**
2. Operators
3. Genotypes and phenotypes
4. Experimental methodology

Reminder: How to escape local optima



1. Allow **larger jumps** in the `nbr` function
2. **Restart** (**iterated hill-climbing**)
3. Allow **disimproving moves** (**simulated annealing** and **late-acceptance hill-climbing**)
4. Use **multiple search points** with **information transfer** between them (**genetic algorithm** or GA).

Darwinian evolution

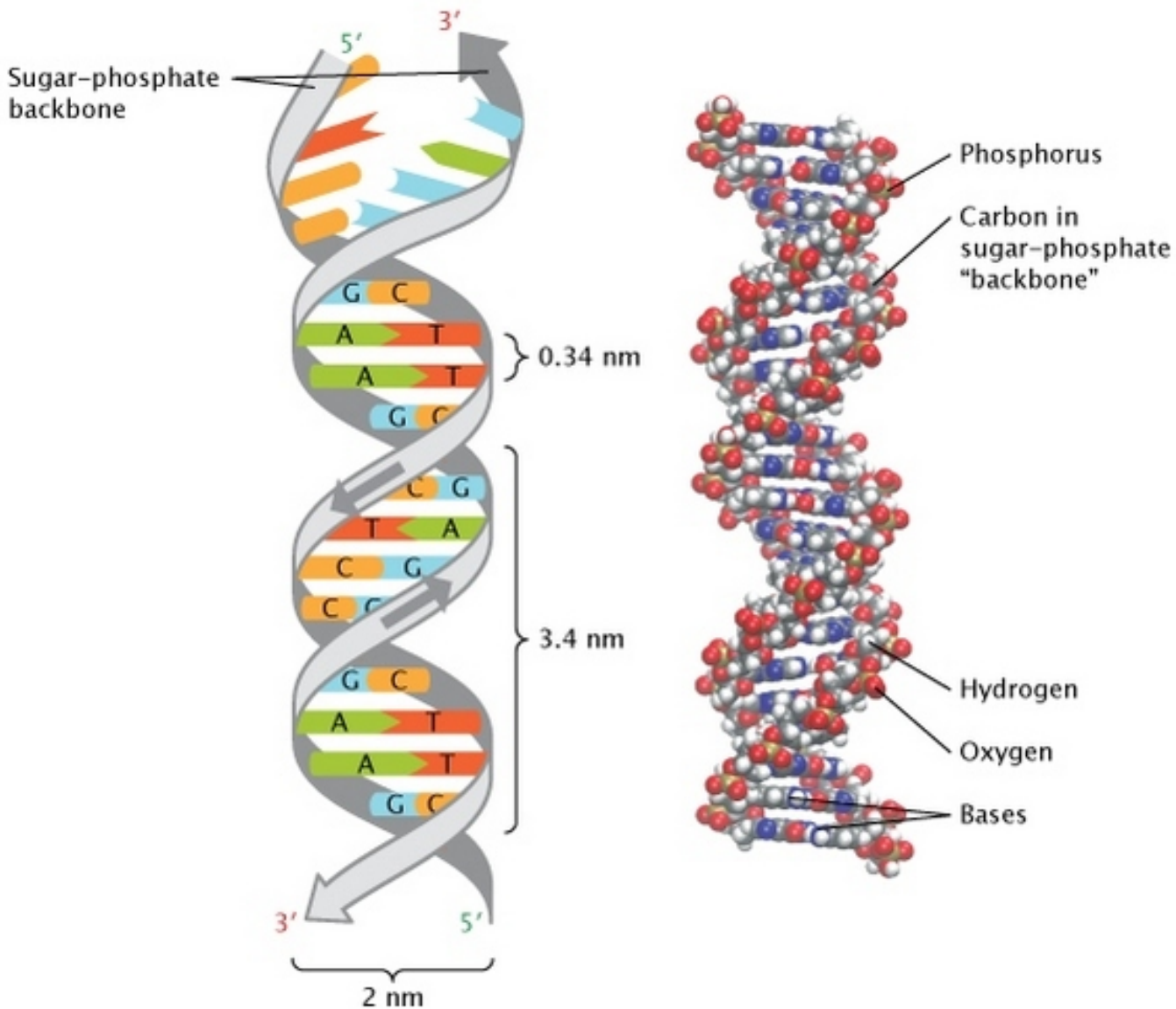


"If I were to give an award for the single best idea anyone ever had, I'd give it to Darwin." – Dennett (1995) "Darwin's Dangerous Idea."

Source: CNN

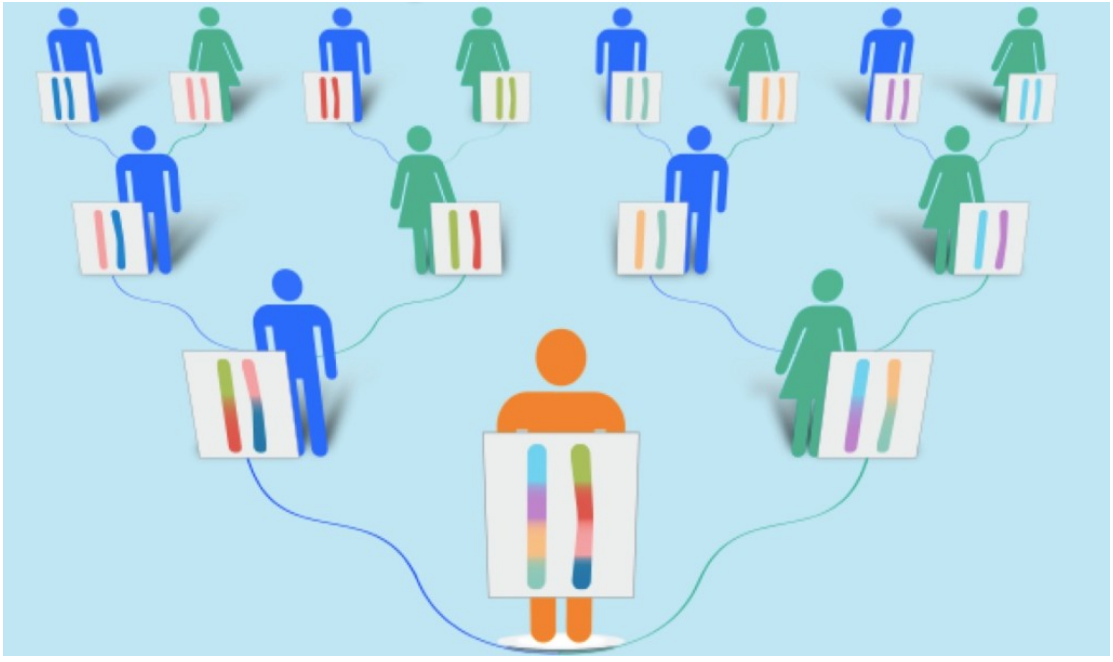
- Population of organisms
- Competition within and between species
- Only the fittest survive and reproduce
- Reproduction: mating and combination of genes
- Mutation of genes
- Inheritance (but no inheritance of acquired traits)

Genes



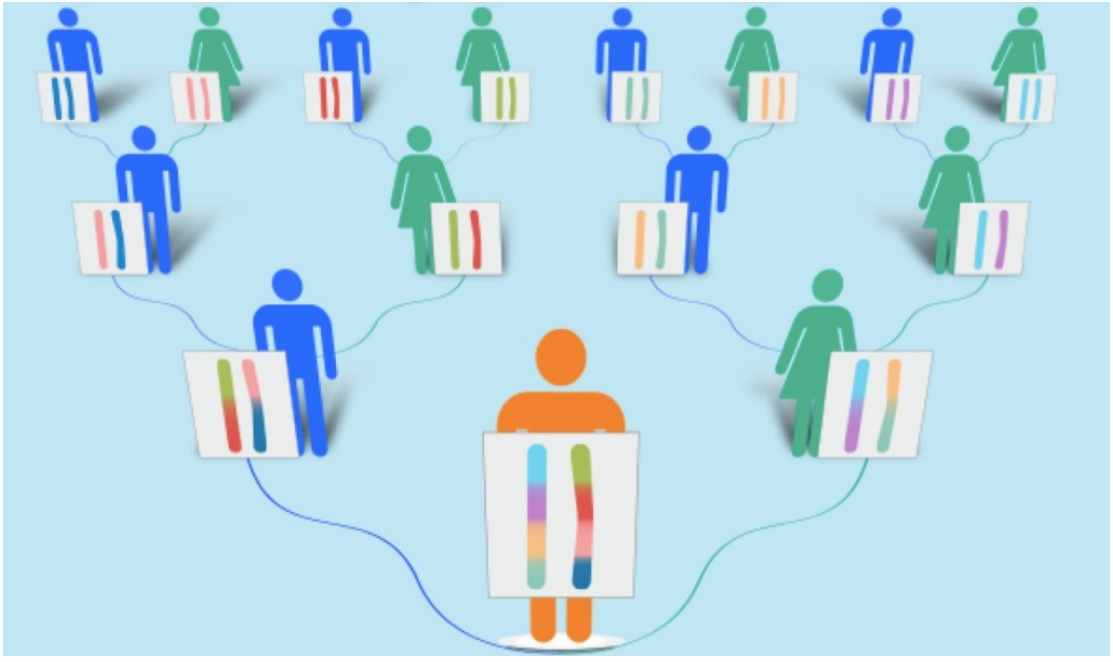
Nature Education 2013

Inheritance and mixing of genes



From regenerationnet

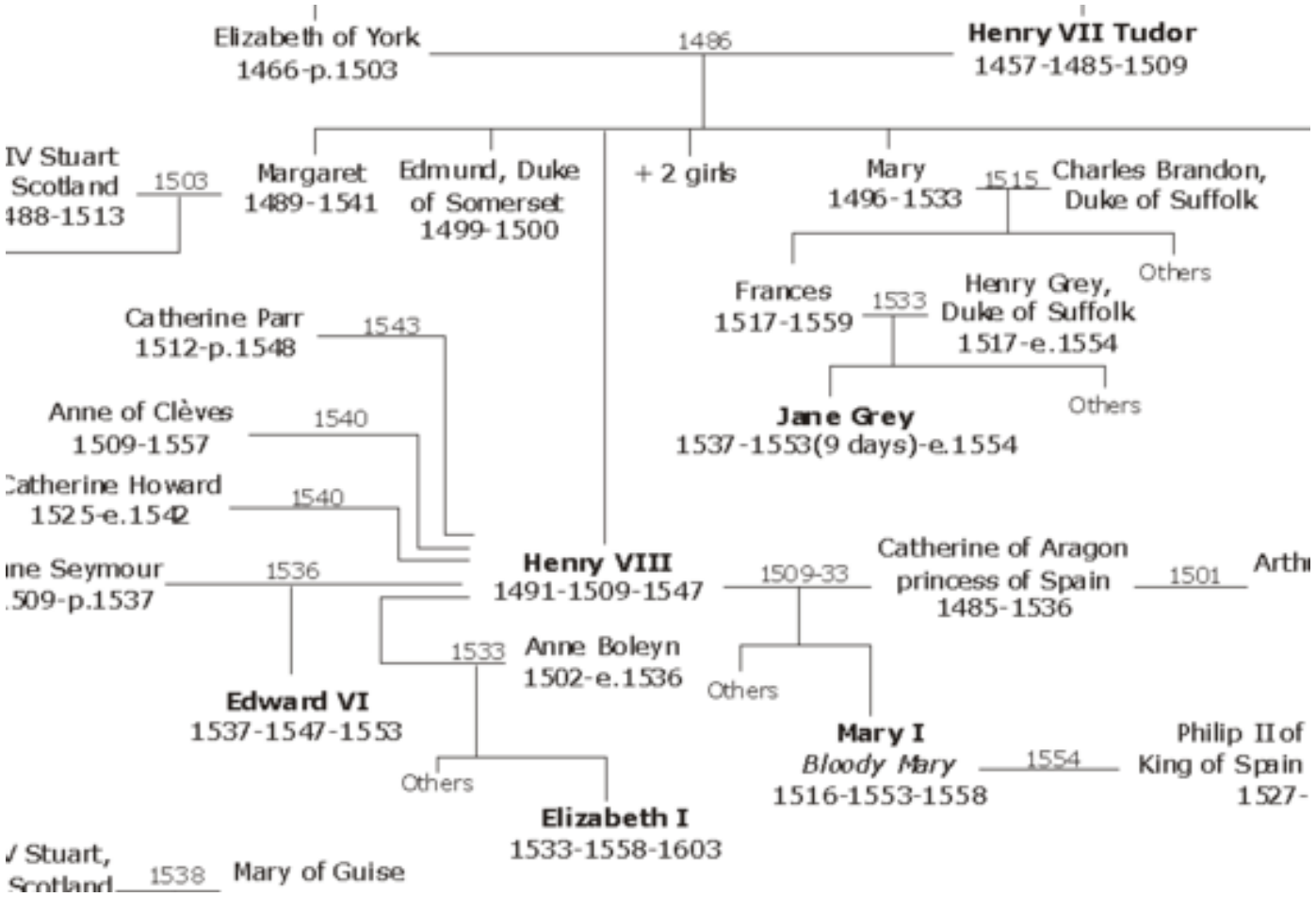
Inheritance and mixing of genes



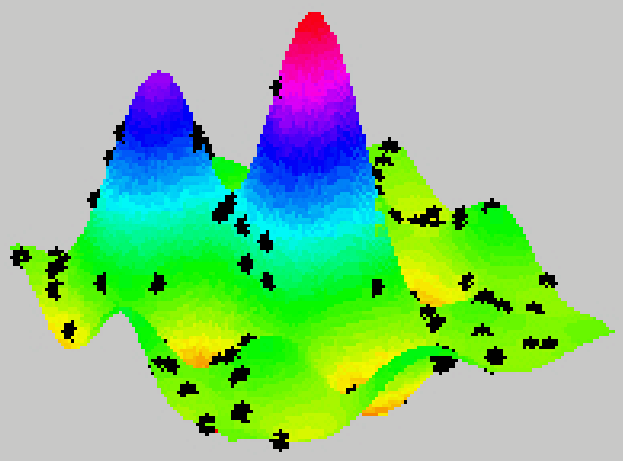
From regenerationnet

How would this tree look if we went back another 100 generations?

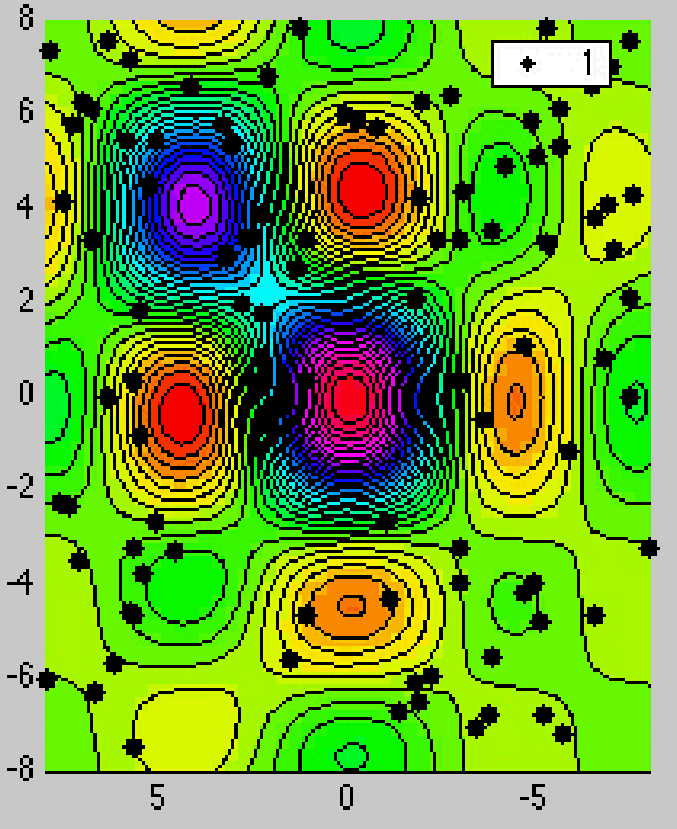
Inheritance and mixing of genes



Population change over time



Tom Cwik JPL/NASA



Evolution = optimisation?

In nature, there is no explicit objective function! No-one is trying to optimise anything.

But there is an **implicit** objective function – ability to survive, compete, and reproduce.

This has the **effect** of optimisation – cheetahs gradually get faster.

Evolutionary algorithms

Evolutionary algorithms are a **family** of black-box metaheuristic optimisation algorithms **inspired by** the main ideas of Darwinian evolution.

Evolutionary algorithms

Evolutionary algorithms are a **family** of black-box metaheuristic optimisation algorithms **inspired by** the main ideas of Darwinian evolution.

- There is always a **population** – not just one current point
- There is crossover/recombination – an operator that takes in **two** solutions and outputs one or two **offspring**.
- There are **not** two sexes in the population – any individual can be recombined with any other.

Terminology

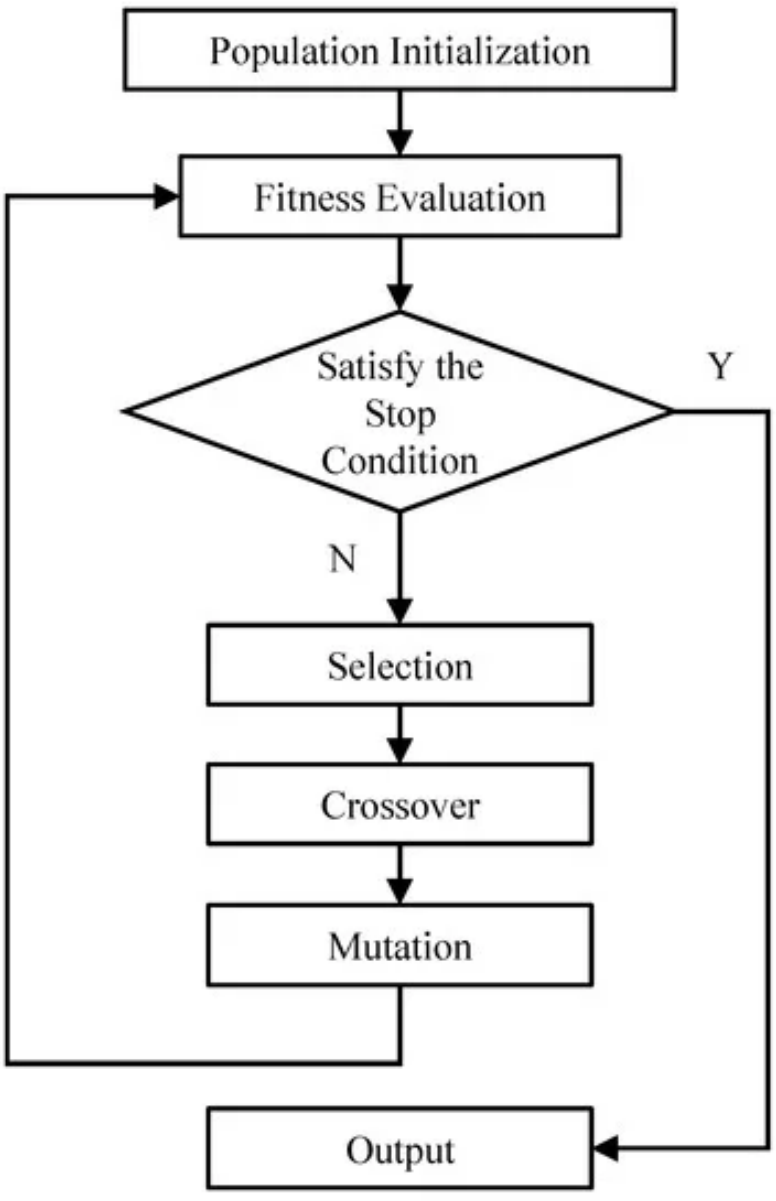
Kirkpatrick

- **Evolutionary Algorithms** or **Evolutionary Computation** is an umbrella term
- The **Genetic Algorithm** (GA) is the central algorithm
- But really, the GA is a huge **family** of variant algorithms
- Sometimes with stupid names like **Intelligent Water-Drop Algorithm**

Terminology

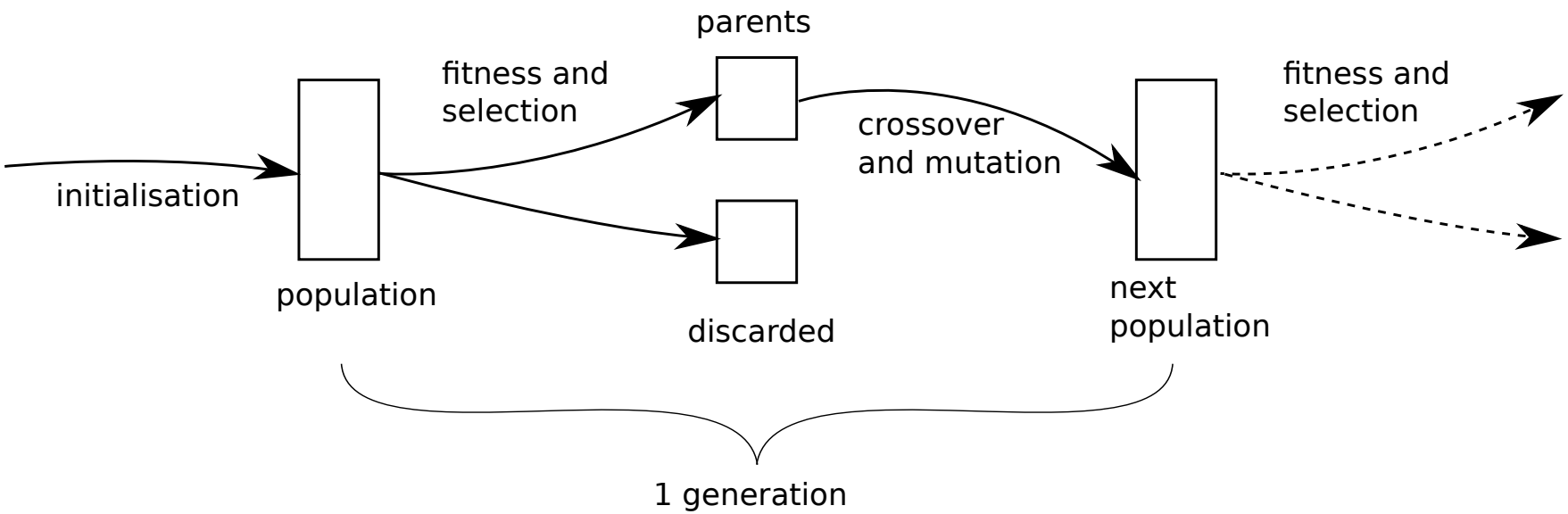
Other optimisation	Evolutionary
Decision variable	Gene
Point/Solution	Genotype/Individual
Multiple points	Population
Objective	Fitness
Neighbour	Mutation
(Combination)	Crossover
Iteration	Generation

GA flowchart



From Xiang et al

GA loop


parents
fitness and
selection
fitness and
selection
crossover
and mutation
initialisation
population
discarded
next
population
1 generation

Overview

Operators

GAs need a few main operators:

- Genetic operators
  - Initialisation
  - Mutation
  - Crossover
- Selection
- (Elitism)

Operators

GAs need a few main operators:

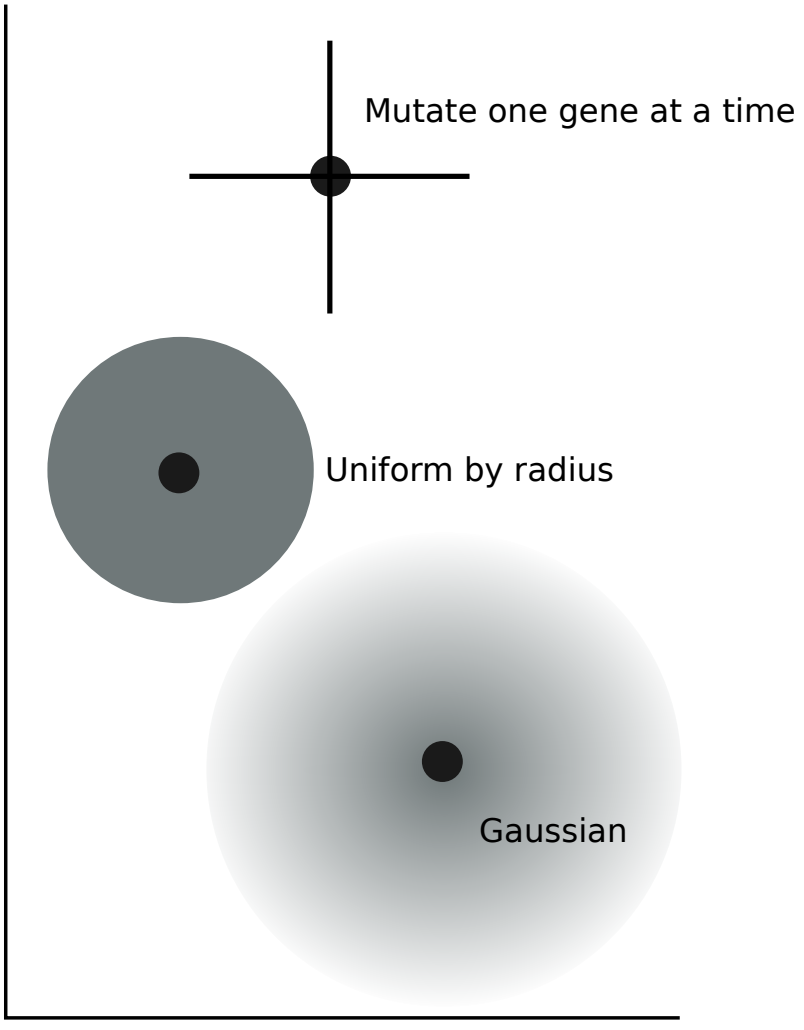
- Genetic operators
  - Initialisation
  - Mutation
  - Crossover
- Selection
- (Elitism)

For each of these, there are many possible ways to design and implement the operator.

Initialisation

- We create a population of n individuals
- Usually **uniformly distributed** in the search space
- `[init() for i in range(n)]`

Mutation



- **Mutation = neighbour**
- May be many possible mutation functions for a given search space, with different properties
- E.g. some more explorative, some more exploitative
- E.g. Gaussian mutation versus uniform mutation
- E.g. mutate one gene chosen randomly, versus mutate all genes with a certain low probability, versus mutate all genes with a small step-size.

Crossover

Crossover is the means of **information transfer** between individuals.

Crossover is the main feature that distinguishes a GA from a HC method.

Crossover is usually seen as **more important** than mutation. Mutation may be omitted or applied to a small proportion of the population.

Crossover

Crossover is the means of **information transfer** between individuals.

Crossover is the main feature that distinguishes a GA from a HC method.

Crossover is usually seen as **more important** than mutation. Mutation may be omitted or applied to a small proportion of the population.

(The other feature is a **population** as opposed to a single current point, but crossover **implies** a population!)

Crossover

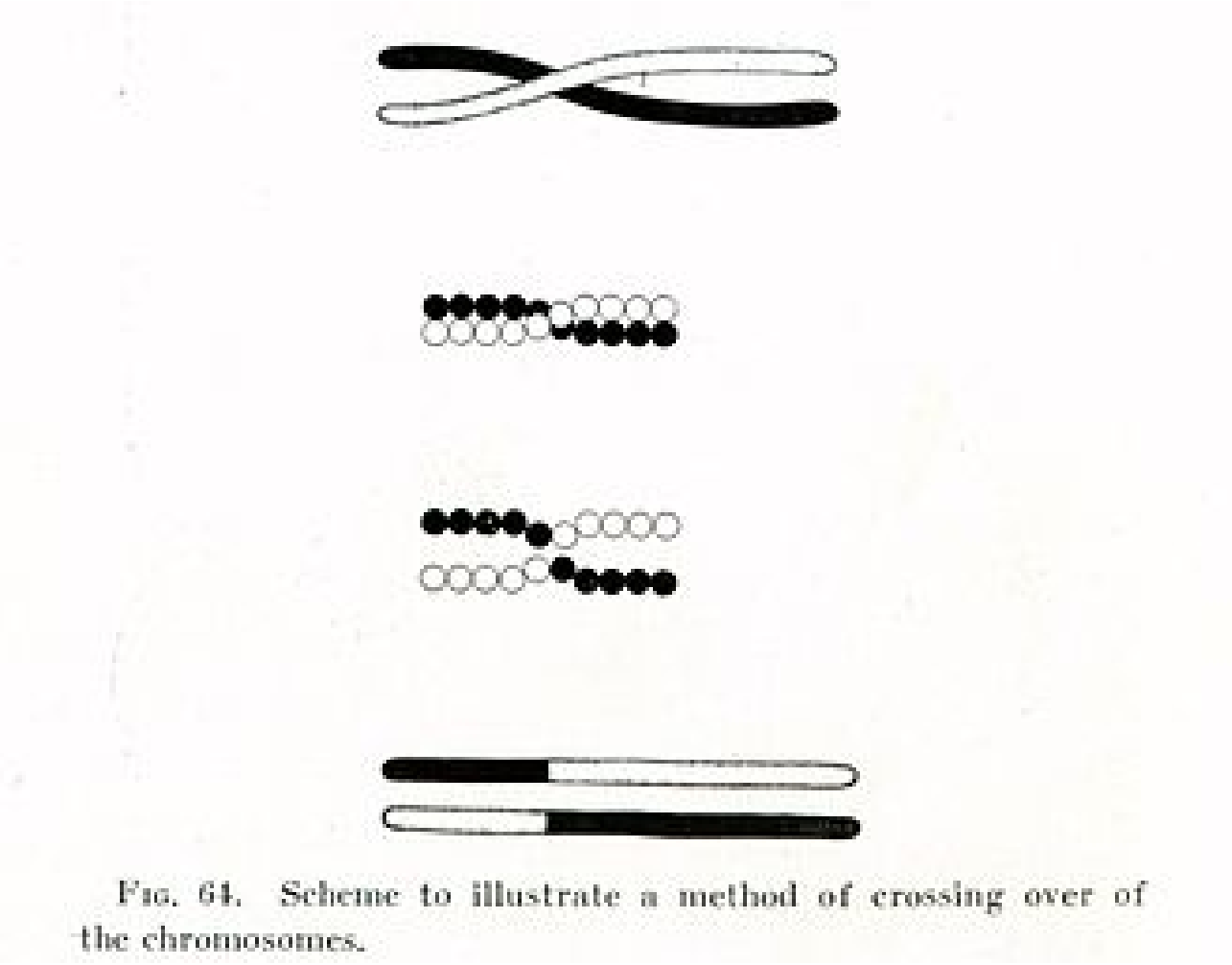


FIG. 64. Scheme to illustrate a method of crossing over of the chromosomes.

Thomas Hunt Morgan, 1916, taken from Wiki

- Each individual's **genome** is a **vector** of DVs.

Crossover

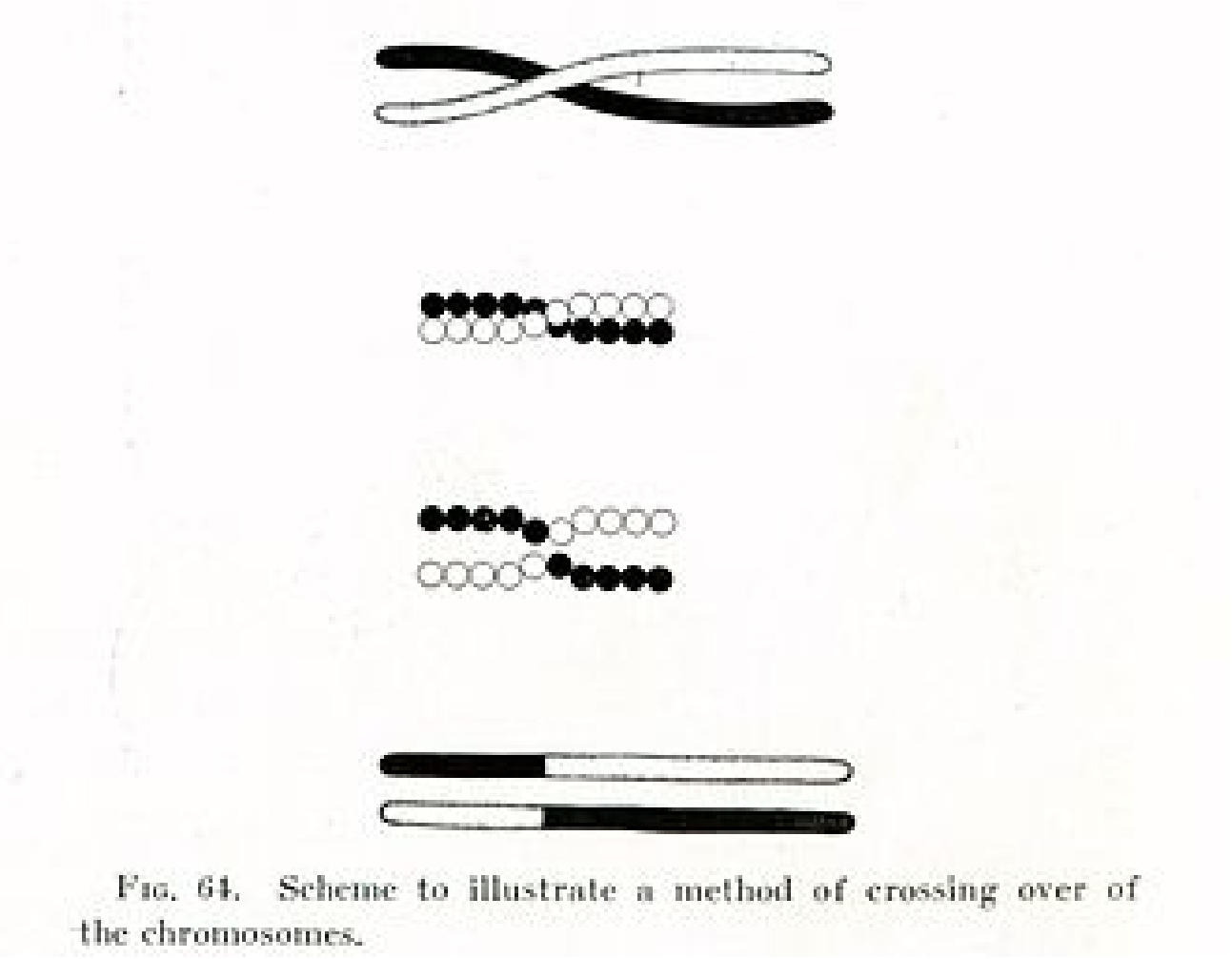
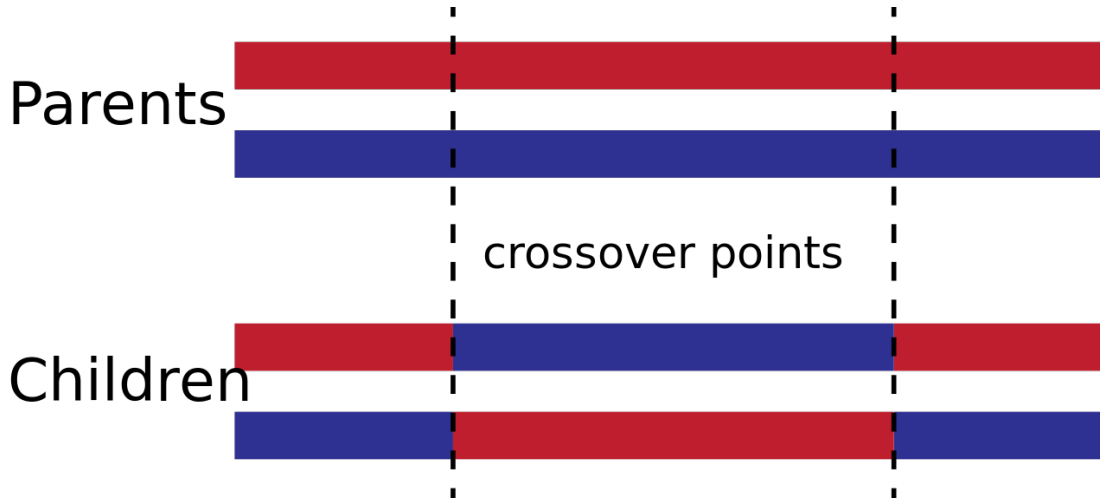


FIG. 64. Scheme to illustrate a method of crossing over of the chromosomes.

Thomas Hunt Morgan, 1916, taken from Wiki

- Each individual's **genome** is a **vector** of DVs.
- The scheme shown here is called **one-point** crossover because we choose one (random) split-point.

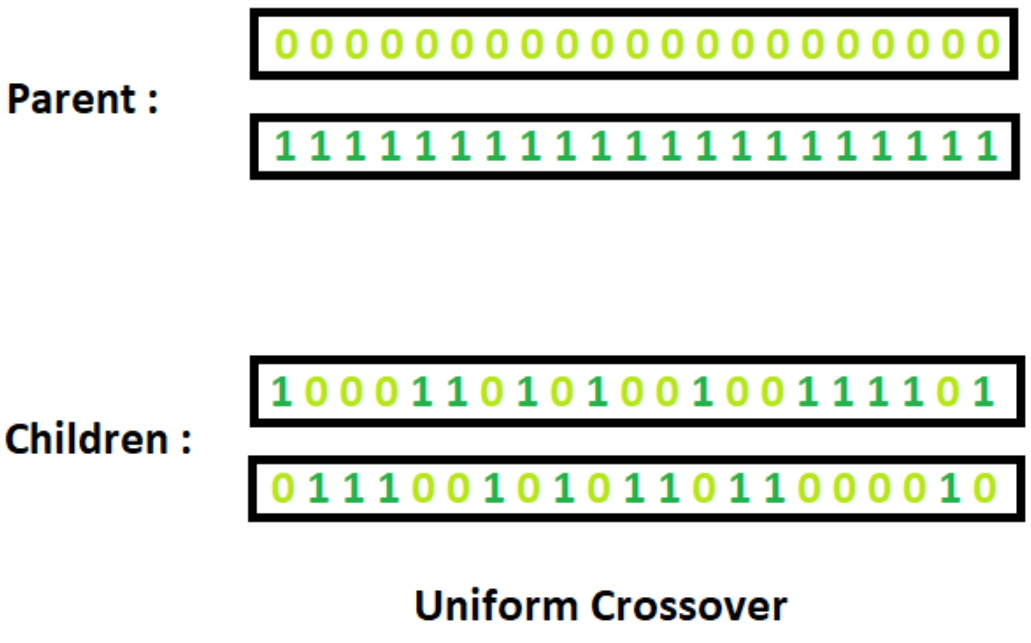
Two-point crossover



This is **two-point** because we choose two split-points at random locations as shown.

Uniform crossover

Every position gets a value chosen from the same position in one parent or the other



Geeks for geeks

Uniform crossover

```
def uniform_crossover(x, y):
    c, d = [], []
    for xi, yi in zip(x, y):
        if random.random() < 0.5:
            c.append(xi); d.append(yi)
        else:
            c.append(yi); d.append(xi)
    return c, d
```

Uniform crossover

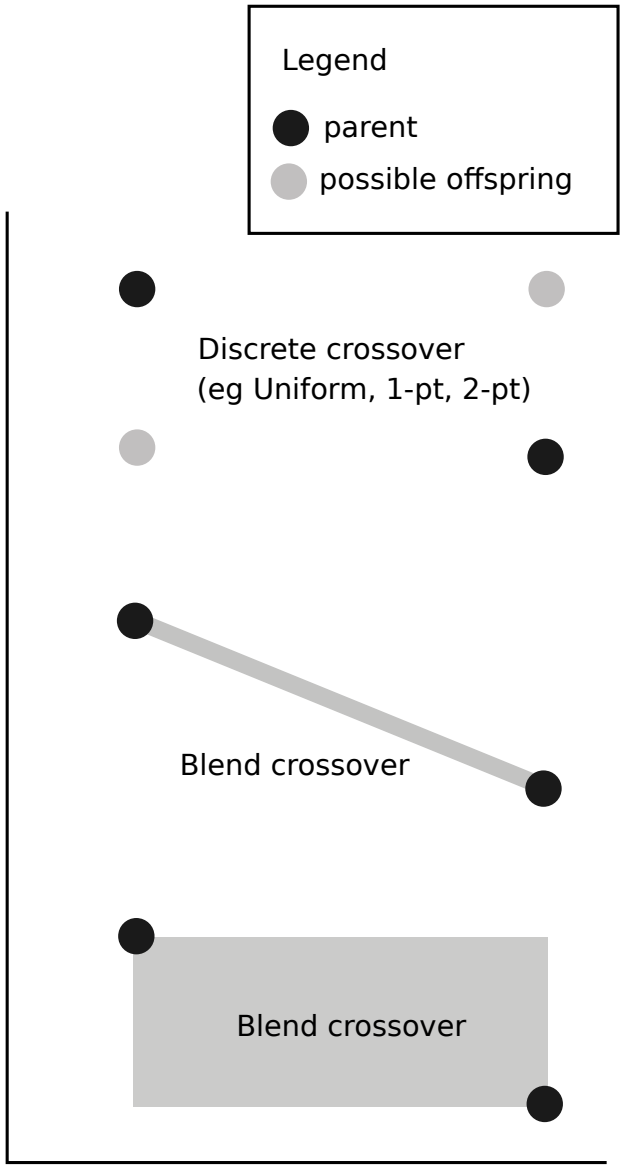
```
def uniform_crossover(x, y):
    c, d = [], []
    for xi, yi in zip(x, y):
        if random.random() < 0.5:
            c.append(xi); d.append(yi)
        else:
            c.append(yi); d.append(xi)
    return c, d
```

You can easily implement any of the crossover operators we have seen.

Crossover offspring

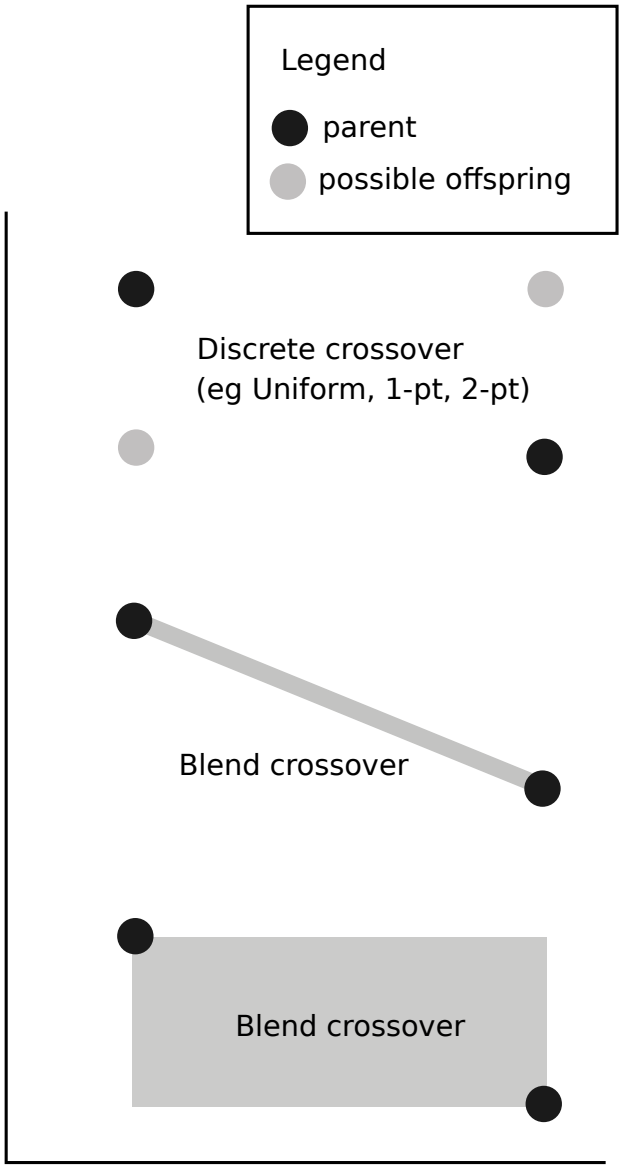
Some operators are defined to return just one offspring; some return two.

Crossover: where do offspring go?



- Offspring are usually somehow **intermediate** to their parents.

Crossover: where do offspring go?



- Offspring are usually somehow **intermediate** to their parents.
- This has important implications! Crossover can only search "within" the area covered by the population (for some definition of "within"). A little mutation is needed to go outside it sometimes.

Linkage between DVs

- Recall the California Manufacturing Co. factory/warehouse IP problem: we can only build a warehouse in LA if we also build a factory there
- That is a **dependency** between two DVs (genes)
- We implemented it as a **constraint**

Linkage between DVs

- Recall the California Manufacturing Co. factory/warehouse IP problem: we can only build a warehouse in LA if we also build a factory there
- That is a **dependency** between two DVs (genes)
- We implemented it as a **constraint**

- Recall Week 05 lab we constructed a harder bitstring problem by **multiplying** two DVs.

Linkage between DVs

- Recall the California Manufacturing Co. factory/warehouse IP problem: we can only build a warehouse in LA if we also build a factory there
- That is a **dependency** between two DVs (genes)
- We implemented it as a **constraint**
- Recall Week 05 lab we constructed a harder bitstring problem by **multiplying** two DVs.
- **Linkage**: dependence in the **objective** function between multiple DVs

Linkage and building-blocks

- We hope the GA will assemble good genetic **building-blocks** – sub-sequences of genes which work well together

Linkage and building-blocks

- We hope the GA will assemble good genetic **building-blocks** – sub-sequences of genes which work well together
- We hope that crossover will tend to preserve good building-blocks, not destroy them.

Linkage and building-blocks

- We hope the GA will assemble good genetic **building-blocks** – sub-sequences of genes which work well together
- We hope that crossover will tend to preserve good building-blocks, not destroy them.
- Both one-point and two-point crossover tend to keep **chunks** of the genome together

Linkage and building-blocks

- We hope the GA will assemble good genetic **building-blocks** – sub-sequences of genes which work well together
- We hope that crossover will tend to preserve good building-blocks, not destroy them.
- Both one-point and two-point crossover tend to keep **chunks** of the genome together
- Uniform crossover does not

Linkage and building-blocks

- We hope the GA will assemble good genetic **building-blocks** – sub-sequences of genes which work well together
- We hope that crossover will tend to preserve good building-blocks, not destroy them.
- Both one-point and two-point crossover tend to keep **chunks** of the genome together
- Uniform crossover does not
- But are related genes always nearby on the genome?

Linkage and building-blocks

Are related genes always nearby on the genome? Consider these two definitions of the warehouse problem genome (all variables binary):

```
[LA_warehouse, LA_factory, SF_warehouse, SF_factory]
```

versus

```
[LA_warehouse, SF_warehouse, LA_factory, SF_factory]
```

Linkage and building-blocks

Are related genes always nearby on the genome? Consider these two definitions of the warehouse problem genome (all variables binary):

```
[LA_warehouse, LA_factory, SF_warehouse, SF_factory]
```

versus

```
[LA_warehouse, SF_warehouse, LA_factory, SF_factory]
```

The first one keeps linked DVs together.

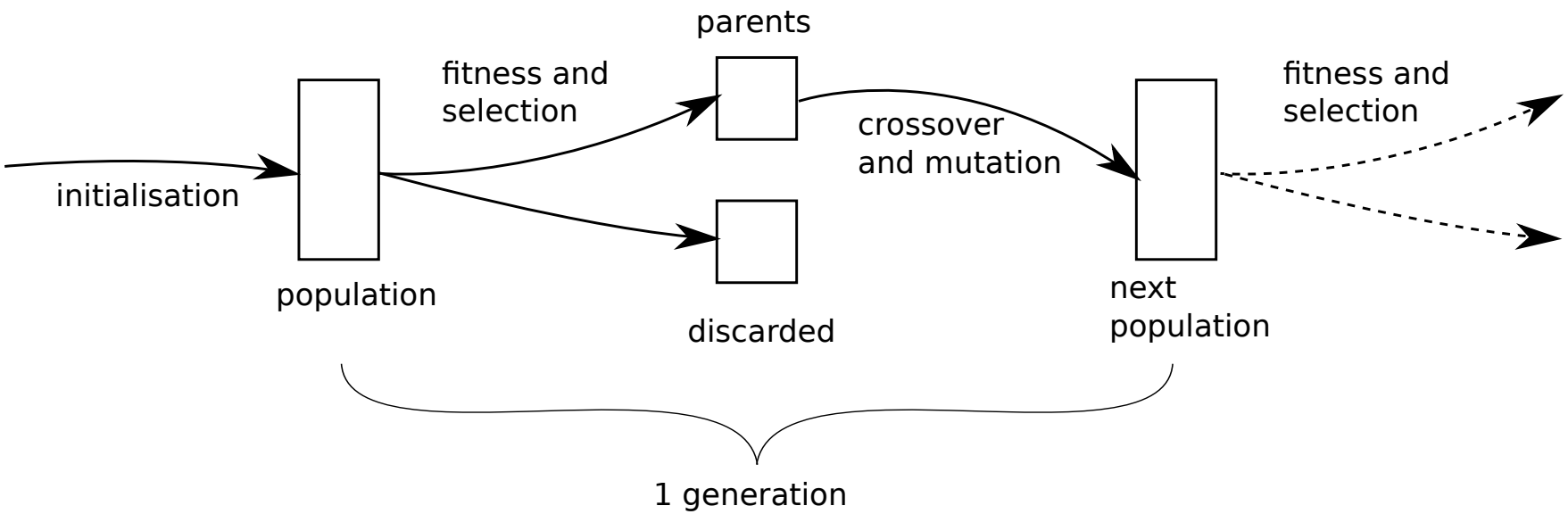
Linkage

"Crossover methods also assume that there is some degree of linkage between genes on the chromosome: that is, settings for certain genes in groups are strongly correlated to fitness improvement. For example, genes A and B might contribute to fitness only when they're both set to 1: if either is set to 0, then the fact that the other is set to 1 doesn't do anything. One- and Two-point Crossover also make the even more tenuous assumption that your vector is structured such that highly linked genes are located near to one another on the vector: because such crossovers are unlikely to break apart closely-located gene groups. Unless you have carefully organized your vector, this assumption is probably a bug, not a feature. Uniform Crossover also makes some linkage assumptions but does not have this linkage-location bias. Is the general linkage assumption true for your problem? Or are your genes essentially independent of one another? For most problems of interest, it's the former: but it's dicey. Be careful." – Luke, **Essentials**.

Genetic operators' desirable properties

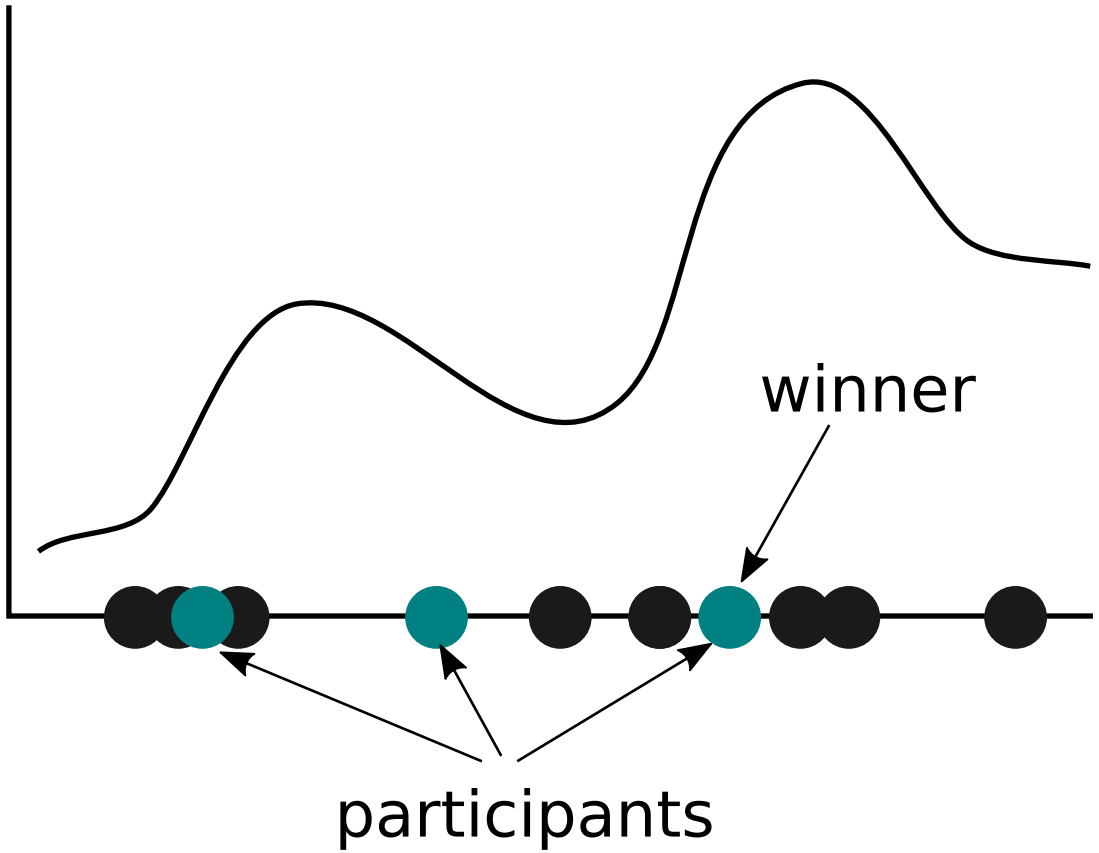
- Initialisation:
  - Uniform on the space
- Crossover:
  - Offspring are intermediate to parents
  - Undirected
  - Allow building blocks if they exist
- Mutation:
  - Mutated genotype is close to original
  - **Ergodic**: capable of reaching any point in the space (through many mutations)
  - Undirected

GA loop


parents
fitness and
selection
fitness and
selection
crossover
and mutation
initialisation
population
discarded
next
population
1 generation

Tournament selection

```
def tournament_select(pop, size):
    return max(random.sample(pop, size), key=f)
```



Tournament selection

Notice that the winner is not the very best individual in the population! (But maybe it will win another tournament later in the same generation.) We don't want the very best individual to win always – it would be too **exploitative** and would cause the algorithm to **converge** immediately.

Tournament selection

Notice that the winner is not the very best individual in the population! (But maybe it will win another tournament later in the same generation.) We don't want the very best individual to win always – it would be too **exploitative** and would cause the algorithm to **converge** immediately.

Larger tournament sizes give a **higher selection pressure**, thus make the algorithm **more exploitative**.

Tournament selection

Notice that the winner is not the very best individual in the population! (But maybe it will win another tournament later in the same generation.) We don't want the very best individual to win always – it would be too **exploitative** and would cause the algorithm to **converge** immediately.

Larger tournament sizes give a **higher selection pressure**, thus make the algorithm **more exploitative**.

What would happen if the tournament size equalled the population size?

Tournament selection

Tournament selection is nice because it is robust and tunable (exploration versus exploitation). Some other methods are less robust, e.g. the **roulette wheel** can be too exploitative early on and too explorative later in the search.

Why do GAs work?

“Consider everything. Keep the good. Avoid evil whenever you notice it.’ ’ (1 Thess. 5:21-22)

(quoted on https://www.mat.univie.ac.at/~neum/glopt.html)

Why do GAs work?

"Consider everything. Keep the good. Avoid evil whenever you notice it.' ' (1 Thess. 5:21-22)

(quoted on https://www.mat.univie.ac.at/~neum/glopt.html)

Just like with hill-climbing, the **genetic operators** (mutation and crossover) are **undirected**, unaffected by fitness. It is **selection** and the **ratchet** which "drive" evolution.

Why do GAs work?

"Consider everything. Keep the good. Avoid evil whenever you notice it.' ' (1 Thess. 5:21-22)

(quoted on https://www.mat.univie.ac.at/~neum/glopt.html)

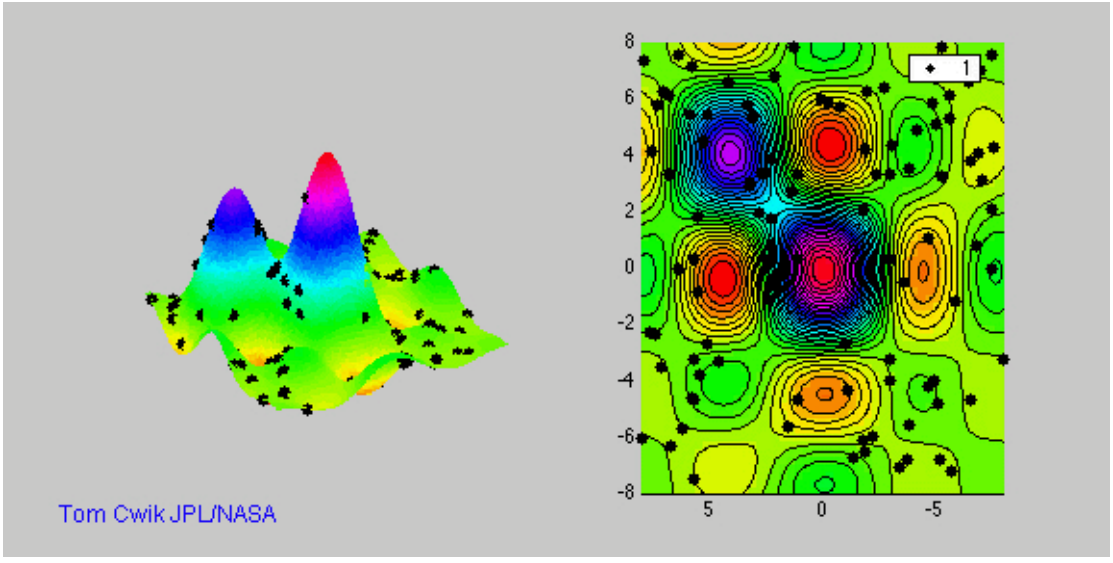
Just like with hill-climbing, the **genetic operators** (mutation and crossover) are **undirected**, unaffected by fitness. It is **selection** and the **ratchet** which "drive" evolution.

Recall the **ratchet** in hill-climbing. Can you see how to rewrite hill-climbing using **selection** in place of the `if`-statement?

Elitism

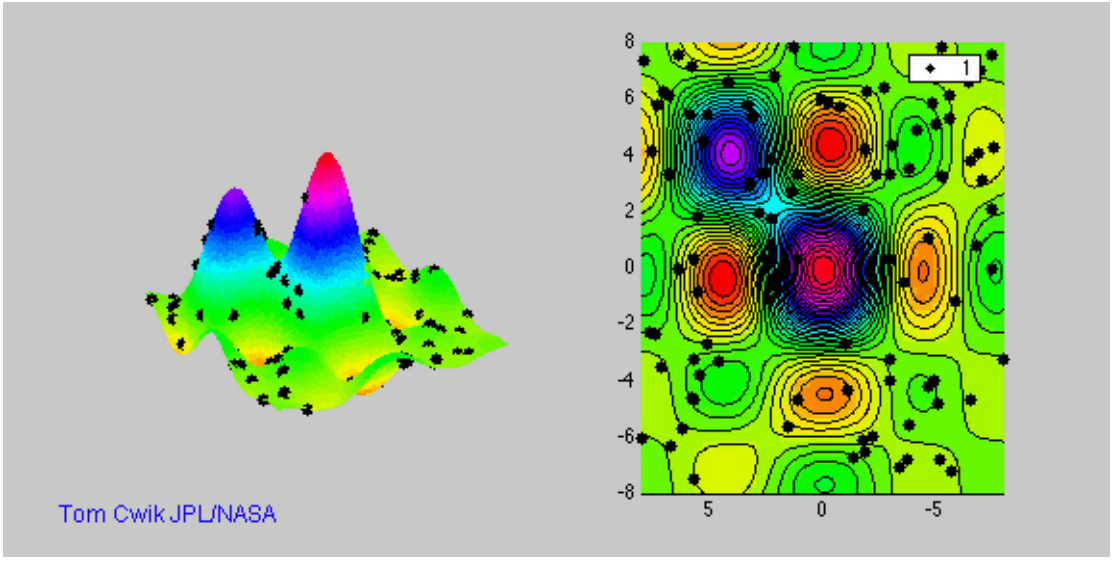
- In hill-climbing, we allow disimproving moves to help escape local optima
- In the GA, we don't have to discard our best individual to escape
- No real need to allow a disimprovement in the **best** objective value from one generation to the next
- **Elitism** means: we copy the best individual in the population directly to the next generation
- (sometimes more than one).

Convergence



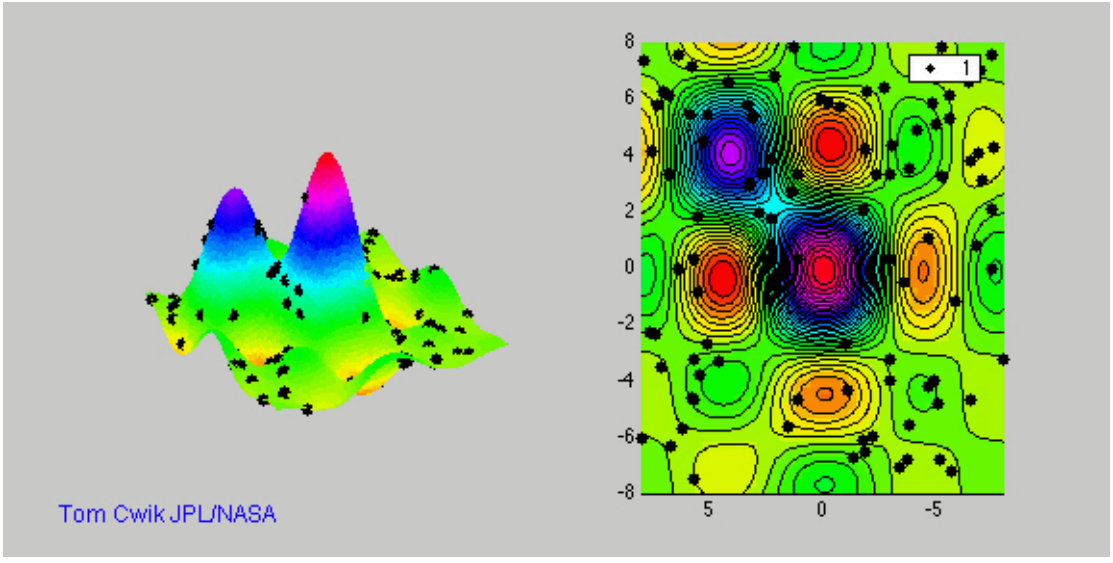
- We can measure genetic diversity in the population

Convergence



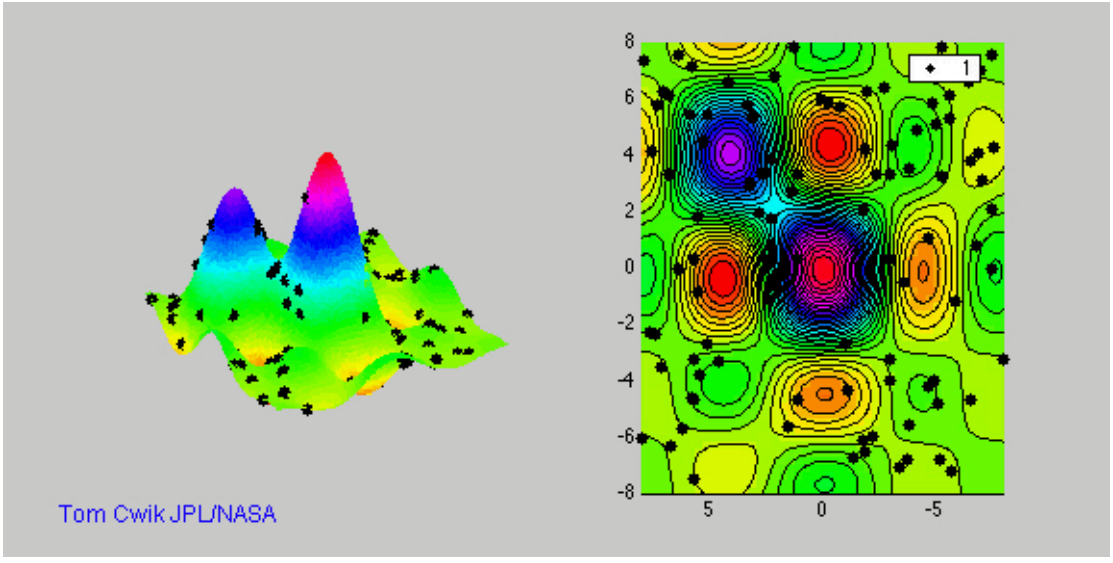
- We can measure genetic diversity in the population
- Or measure variance of objective values in the population

Convergence



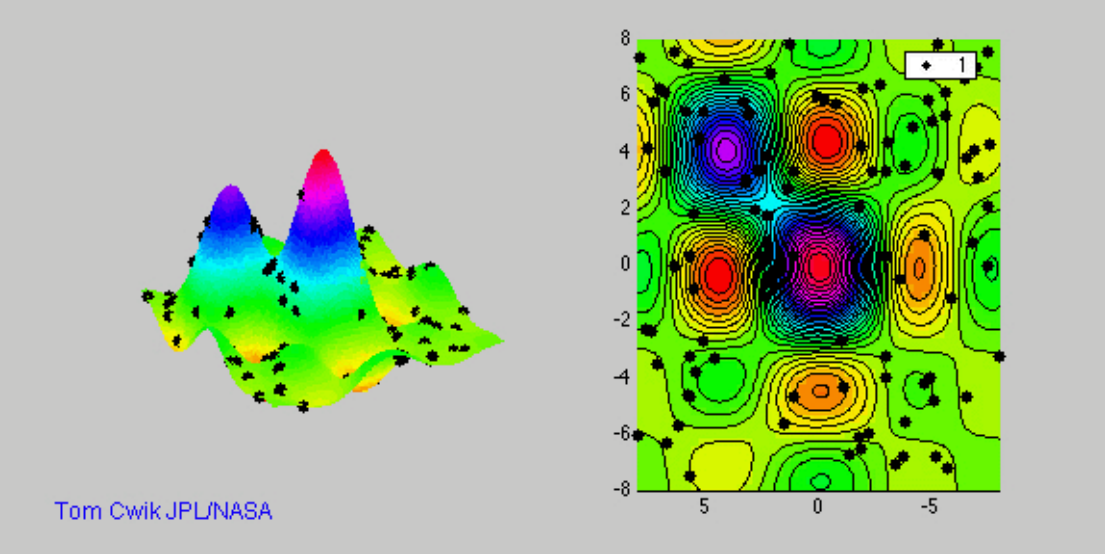
- We can measure genetic diversity in the population
- Or measure variance of objective values in the population
- Diversity **decreases over time**

Convergence



- We can measure genetic diversity in the population
- Or measure variance of objective values in the population
- Diversity **decreases over time**
- When the population has **converged**, there will be no more improvement…

Convergence



- We can measure genetic diversity in the population
- Or measure variance of objective values in the population
- Diversity **decreases over time**
- When the population has **converged**, there will be no more improvement…
- … because offspring will be almost identical to parents.

GA assumptions

GAs make the same sorts of assumptions as smart hill-climbing:

- the objective may be a bit rugged, but not totally uninformative or deceptive;

GA assumptions

GAs make the same sorts of assumptions as smart hill-climbing:

- the objective may be a bit rugged, but not totally uninformative or deceptive;
- we don't know the gradient;

GA assumptions

GAs make the same sorts of assumptions as smart hill-climbing:

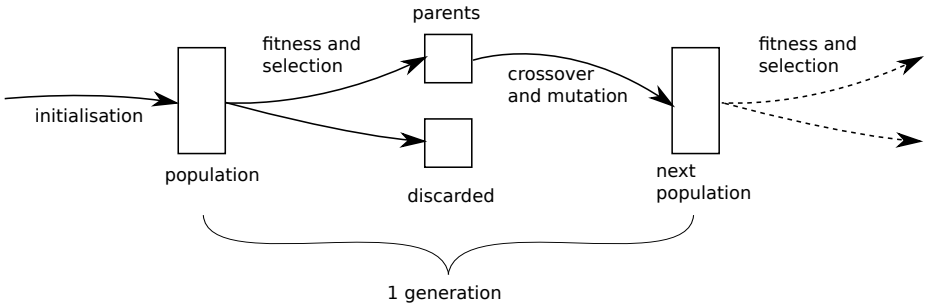
- the objective may be a bit rugged, but not totally uninformative or deceptive;
- we don't know the gradient;
- neighbours (created by mutation) usually have similar fitness;

GA assumptions

GAs make the same sorts of assumptions as smart hill-climbing:

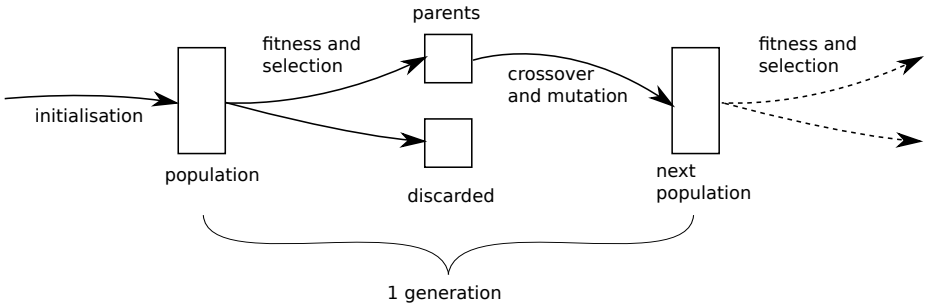
- the objective may be a bit rugged, but not totally uninformative or deceptive;
- we don't know the gradient;
- neighbours (created by mutation) usually have similar fitness;
- children (created by crossover) usually have fitness similar to parents.

GA: alternative loops



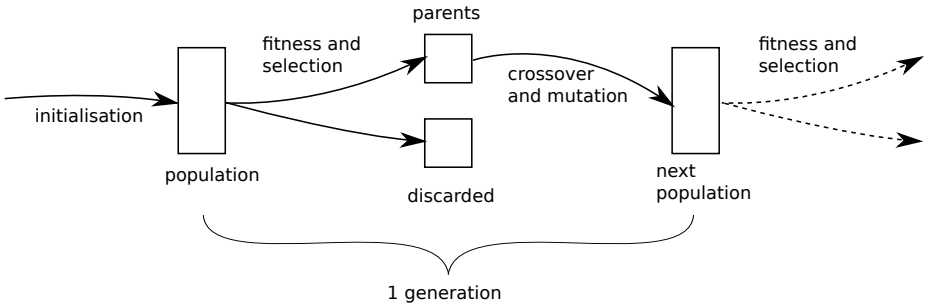
1. E.g. apply crossover to produce some individuals, and mutation to produce others

GA: alternative loops



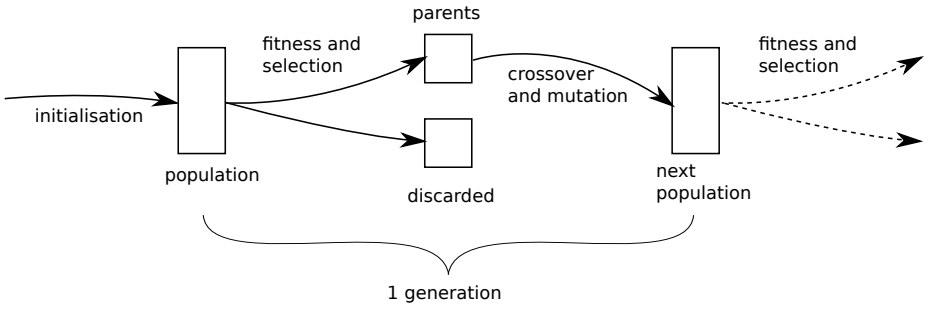
1. E.g. apply crossover to produce some individuals, and mutation to produce others
2. E.g. use entire population to produce offspring, then run fitness/selection to **discard from** (population + offspring)

GA: alternative loops



1. E.g. apply crossover to produce some individuals, and mutation to produce others
2. E.g. use entire population to produce offspring, then run fitness/selection to **discard from** (population + offspring)
3. E.g. "steady-state GA" where we just produce 1 or 2 offspring at a time from any parents, and use fitness/selection to decide which individuals to **discard**. No generations.

GA: alternative loops



1. E.g. apply crossover to produce some individuals, and mutation to produce others
2. E.g. use entire population to produce offspring, then run fitness/selection to **discard from** (population + offspring)
3. E.g. "steady-state GA" where we just produce 1 or 2 offspring at a time from any parents, and use fitness/selection to decide which individuals to **discard**. No generations.
4. E.g. "memetic algorithm", a GA with an inner loop doing local search (e.g. mutation only).

Memetic algorithm

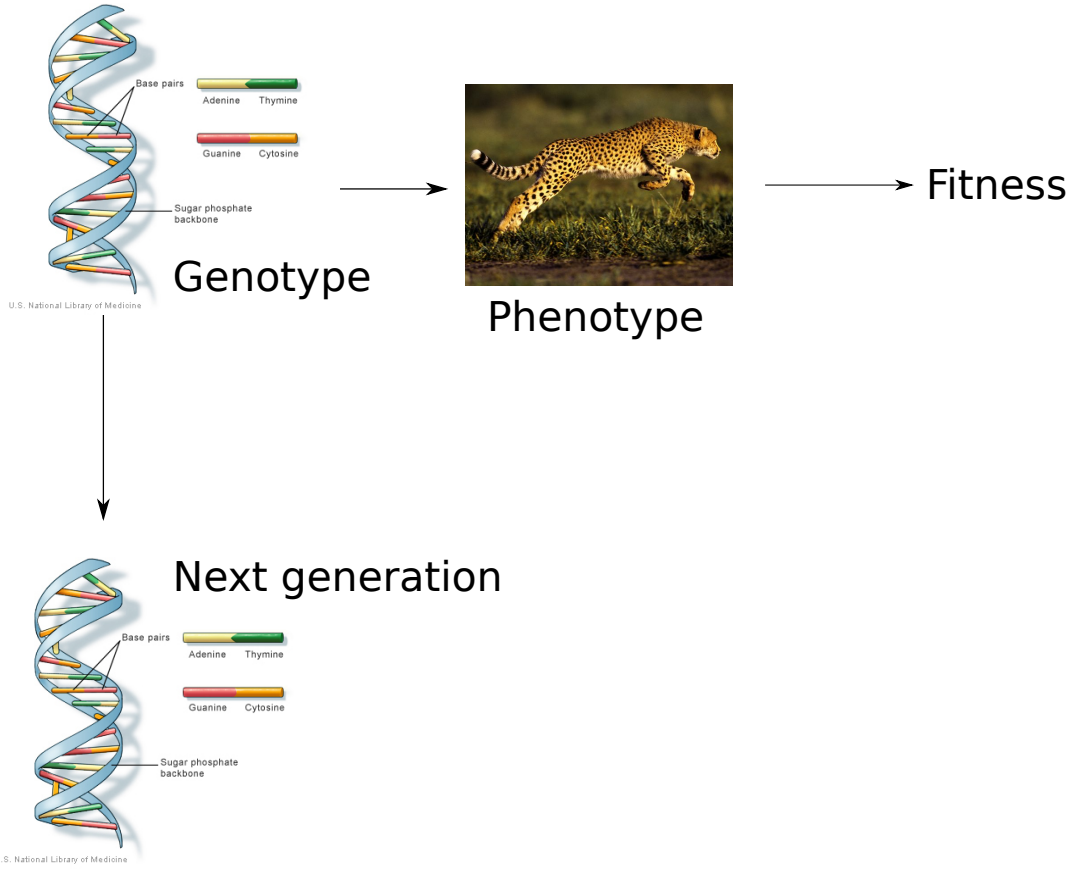
In *The Selfish Gene* (1978), Dawkins proposed that ideas evolve in a way similar to genes, and nicknamed them **memes**.

A **memetic algorithm** is a GA with an inner loop doing local search (e.g. mutation only). This is **not a good name** for the algorithm.


HIPSTER DAWKINS
LIKED MEMES
BEFORE THEY WERE COOL
imgflip.com

Overview

Genotypes and phenotypes



- Mapping from genes (**genotype**) to organism (**phenotype**)
- Only the organism is "evaluated" by nature for its "fitness"; only the genes are passed on to offspring.

Motivation

- Q. **Why** does nature do this?

Motivation

- Q. **Why** does nature do this?
- A. It's impossible to mutate a cheetah, or to crossover two cheetahs. Mutation and crossover work on the underlying DNA, not on the animal.

Example

A car is defined by:

- Shape (8 floats, 1 per vertex)
- Wheel size (2 floats, 1 per wheel)
- Wheel position (2 ints, 1 per wheel)
- Wheel density (2 floats, 1 per wheel) darker means denser
- Chassis density (1 float) darker means denser

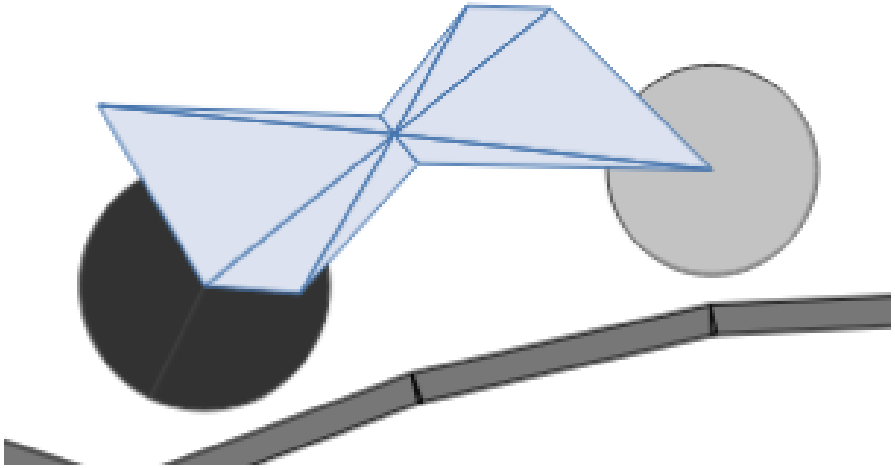


Image from rednuht.org

Phenotype: the car.

Example

A car is defined by:

- Shape (8 floats, 1 per vertex)
- Wheel size (2 floats, 1 per wheel)
- Wheel position (2 ints, 1 per wheel)
- Wheel density (2 floats, 1 per wheel) darker means denser
- Chassis density (1 float) darker means denser

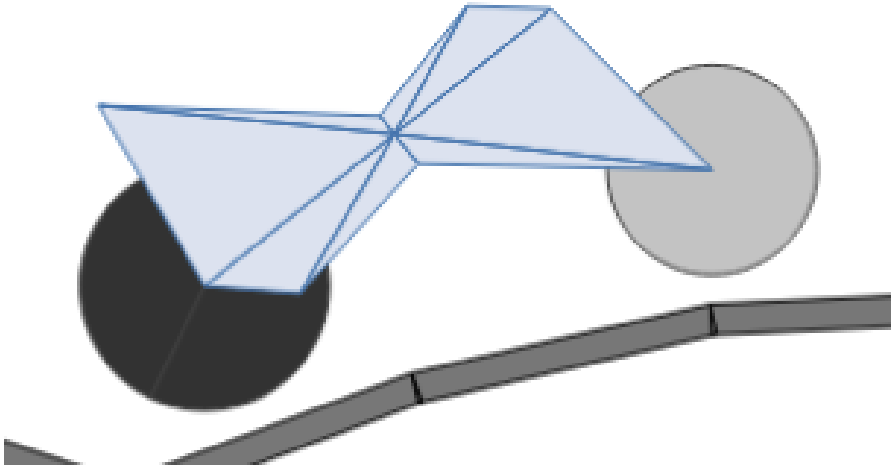


Image from rednuht.org

Phenotype: the car.

Genotype (15 genes): e.g. [.3 .4 .6 .1 .3 .4 .6 .1 | .6 .7 | 3 6 | .3 .8 | .4]

(Full details of genotype-phenotype mapping here.)

Objective: distance travelled in the simulation.

Search space and solution space

The same idea is often used outside Evolutionary Algorithms also, but with different terminology: the **search space** (genotype space) and the **solution space** (phenotype space).

What we actually want is a **solution**, but we run our search in the **search** space, and whenever we look at a search point we map it to a solution.

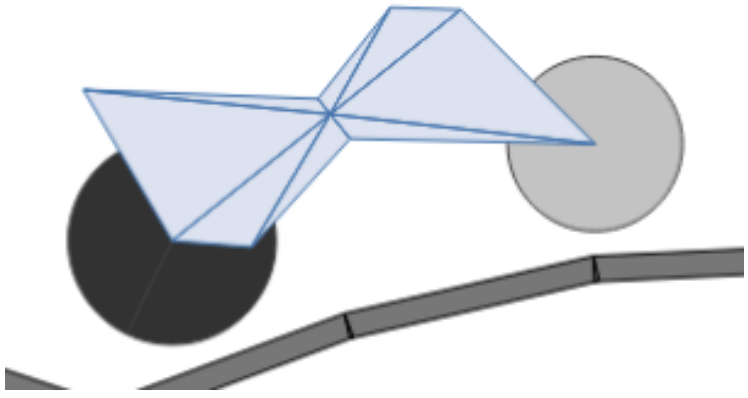
Terminology

Other optimisation	Evolutionary
Decision variable	Gene
Point	Genotype
Search space	Genotype space
Solution space	Phenotype space

Motivation

- Q. **Why** do we do this?
- A. Just like cheetahs, it is not possible to mutate or crossover 2D cars. But it is easy to define these operations on the underlying data structure.

Car genotypes and phenotypes



rednuht.org

The car (phenotype) has to “live” in a simulated world. If it succeeds there, its **genes** are passed on, i.e. are varied for use in the next iteration.

Operators on car genotypes

How could we define these operators on car genotypes?

- Initialisation
- Mutation
- Crossover

Recall that a car genotype is defined as a list of 15 numbers:

- Shape (8 floats, 1 per vertex)
- Wheel size (2 floats, 1 per wheel)
- Wheel position (2 ints, 1 per wheel)
- Wheel density (2 floats, 1 per wheel) darker wheels mean denser wheels
- Chassis density (1 float) darker body means denser chassis

Simulation

The 2D car is also a good example of **simulation**. We don't have a formula or program that directly calculates fitness. The genes give rise to a car (phenotype), and it has to "live" in a simulated world. We measure its success in that world.

Explicit and implicit fitness

Explicit fitness (objective):

- As we saw in LP/IP
- A **formula** or function that gives a **number**

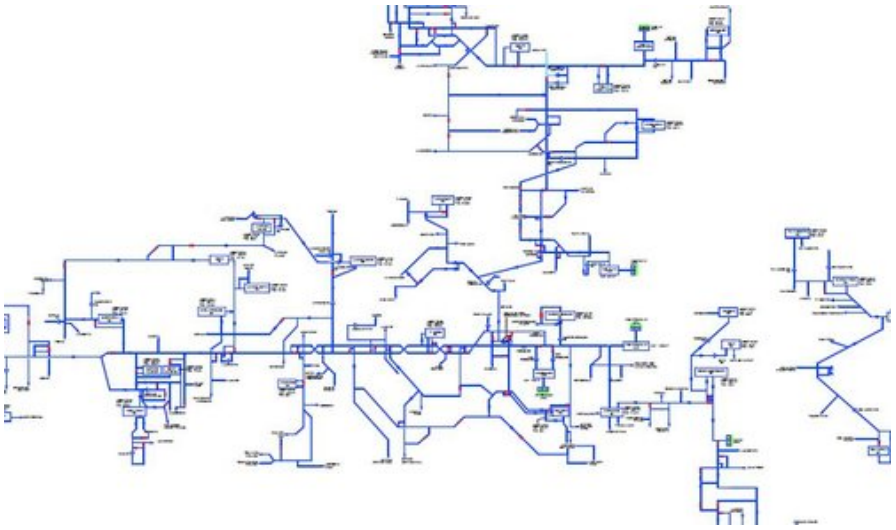
Simulation fitness:

- As in 2D Boxcar
- The solution is put in a simulation, and we somehow measure performance as a **number**

Implicit fitness:

- As in biological evolution
- There is **no number**!
- Research fields: Coevolution and Artificial Life

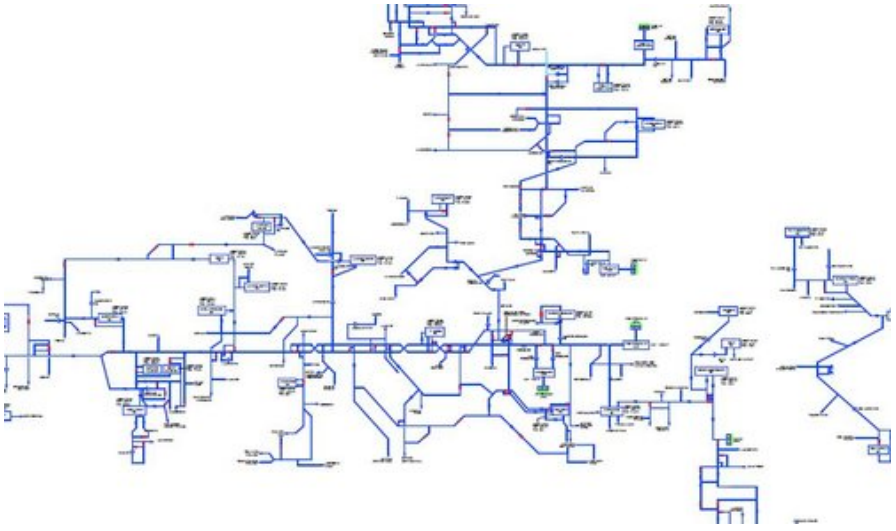
Genotypes and phenotypes: another example



Dwr Uisce

- We want to design a water network (a graph)

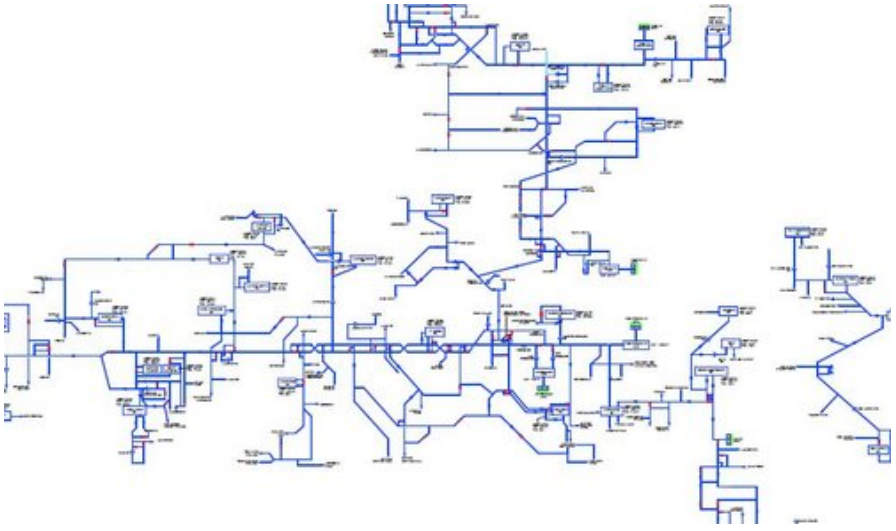
Genotypes and phenotypes: another example



Dwr Uisce

- We want to design a water network (a graph)
- It's hard to define initialisation, mutation and crossover

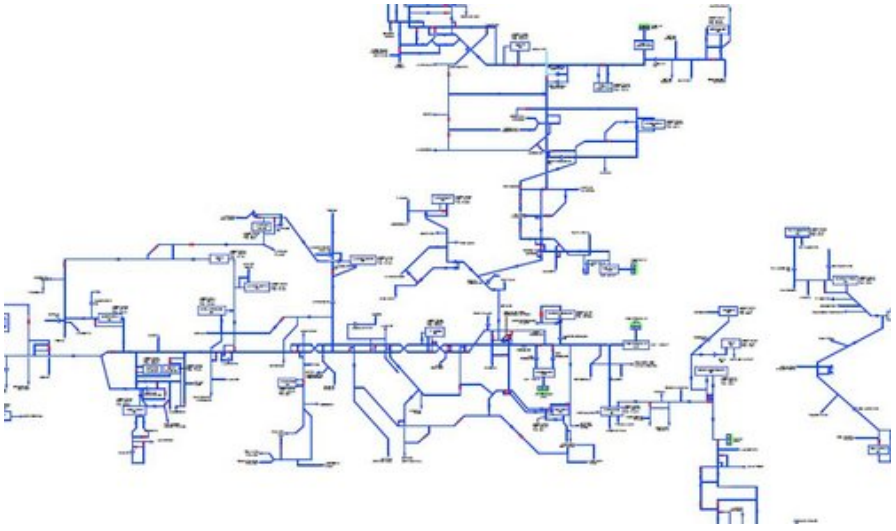
Genotypes and phenotypes: another example



Dwr Uisce

- We want to design a water network (a graph)
- It's hard to define initialisation, mutation and crossover
- Define a mapping from bitstrings (genotypes) to graphs (phenotypes)

Genotypes and phenotypes: another example



Dwr Uisce

- We want to design a water network (a graph)
- It's hard to define initialisation, mutation and crossover
- Define a mapping from bitstrings (genotypes) to graphs (phenotypes)
- Use standard initialisation, mutation and crossover on bitstrings.

Developmental mappings

Above, the mapping from genotype to phenotype was straightforward. Each gene controlled one part of the phenotype directly.

In a **developmental** mapping, it is less straightforward. Multiple genes may influence each part of the phenotype, and one gene may influence multiple parts. That's how it is with real DNA!

We won't explore this further.

Recommended reading: Bentley, **Exploring Component-based Representations - The Secret of Creativity by Evolution?**, in Blackboard.

Overview

1. Introduction to Genetic Algorithms
2. Operators
3. Genotypes and phenotypes
4. **Experimental methodology**

Scientific experiments

When we're trying to solve an optimisation problem, we are free to try anything that works.

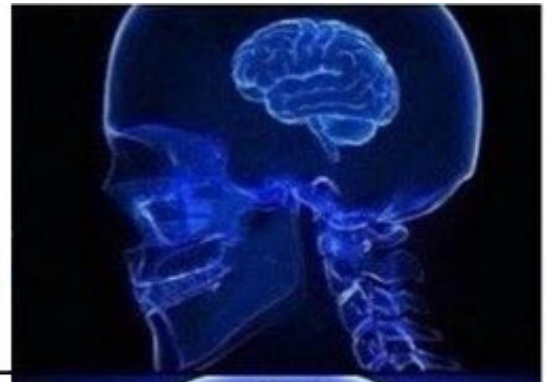
Scientific experiments

When we're trying to solve an optimisation problem, we are free to try anything that works.

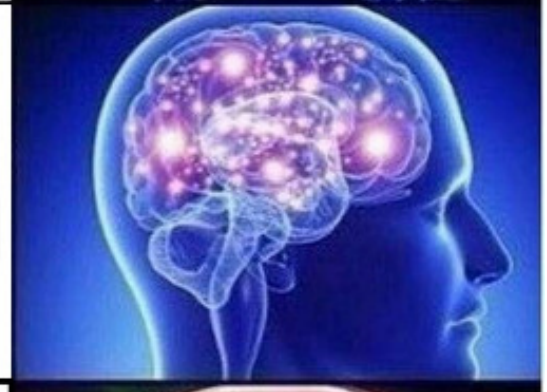
If we want to make **scientific** claims:

- We have a variety of possible algorithms
  - We should compare them
- Our metaheuristics are stochastic (not the same result every time)
  - We should run many times
- Every algorithm has hyper-parameters
  - We should tune them fairly
- Larger fitness evaluation budget implies better performance
  - Use a fixed budget for fairness.

GET A GOOD RESULT FROM OPTIMISATION



OUT-PERFORM RANDOM SEARCH



OUT-PERFORM THE SOTA

OUT-PERFORM THE SOTA AFTER FAIR HYPER-PARAMETER TUNING AND STATISTICAL TESTS

Procedure

1. We identify one or more baselines, including a random search and a **state of the art** method from recent literature.
2. We choose a fixed fitness evaluation budget.
3. We identify the key hyperparameters for each algorithm, and a range of likely values.
4. For each algorithm, for each combination of hyperparameter values, we run the algorithm 30 times.
5. We see which algorithms and configurations have the best **mean objective value**.
6. We carry out an appropriate statistical test (e.g. *t*-test, ANOVA, Kruskal-Wallis, etc.) and we report the *p*-value and whether it is statistically significant at the $\alpha = 0.05$ level.

Random seed

Our algorithms are **randomised**. Running twice with the same hyperparameters may give different results.

It is good practice to set a **random seed** for each run using `random.seed()`. This way our runs are **reproducible**.

```
for seed in range(5):
  random.seed(seed)
  run_my_algorithm()
```

Readings

- Luke, **Essentials**, Section 11.1 (Experimental Methodology): recommended for Assignment 2.