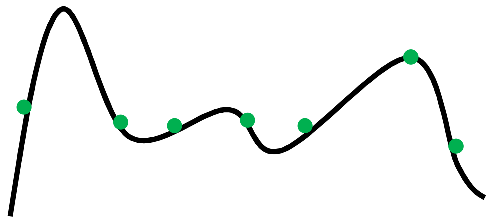
UPPSALA UNIVERSITET

Natural Computation Methods in Machine Learning (NCML)

Lecture 11: Evolutionary Computing 1

Evolutionary Computing

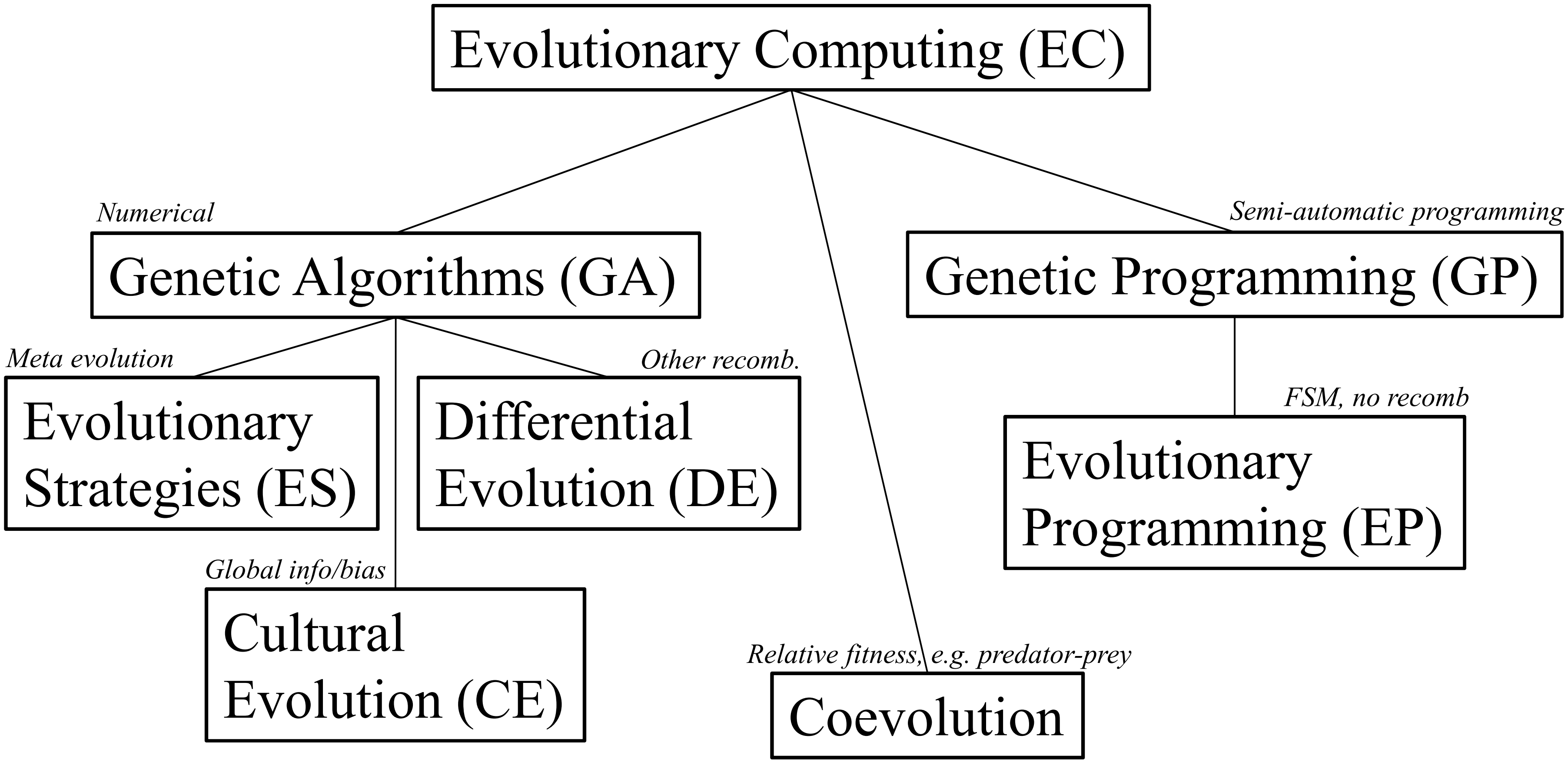
- Population methods
  - Parallel search



- Used for problems where the task is to maximize some measure of success
  - Or to minimize a cost
  - Same family of *problems* as in RL,
    - But very different methods
    - Many individuals, instead of one agent,
    - And they don't move around as in RL
- Methods inspired by genetics, natural selection, evolution

Road Map

Big field. Confusingly many concepts and names

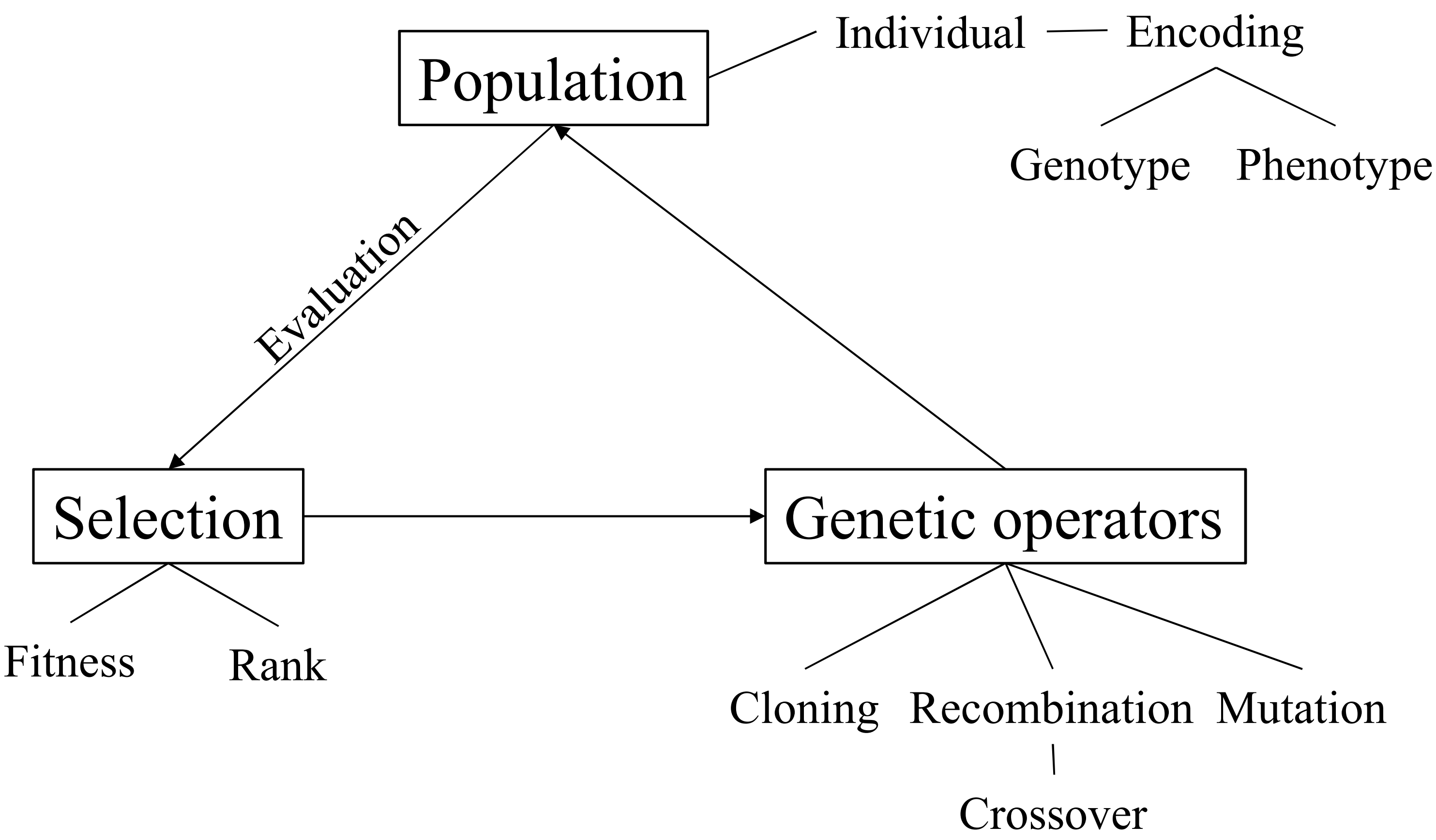


Each with its own chapter in the course book

An evolutionary algorithm
in general

1. Create a population of individuals
   - An individual encodes a suggested problem solution
2. Evaluate all individuals
   - Requires an interpretation of the encoding, and an evaluation of its *fitness*
3. Select individuals
   - Probabilistic, proportional to fitness
4. Create a new population by *cloning*, *recombination* and/or *mutation* of individuals
5. Repeat from step 2 until end criterion met

The Evolutionary Loop



Tip: Avoid the term 'reproduction' (it's ambiguous)

Genotypes, phenotypes and fitness
and a simple Travelling Salesman Problem example

- A solution to the problem is encoded by an individual's *genotype* (*genome*, *chromosome, ...*)
  - For example, in GA it could be a binary string
- TSP example: Visit the 4 oldest Swedish universities, minimizing total distance traveled
  - How to represent a route as a binary string?
  - Naïve approach: Make a table over cities v.s. visit order

	1	2	3	4
Uppsala (1477)	1	0	0	0
Tartu (now Estonian, 1632)	0	0	1	0
Åbo (now Finnish, 1640)	0	1	0	0
Lund (1666)	0	0	0	1

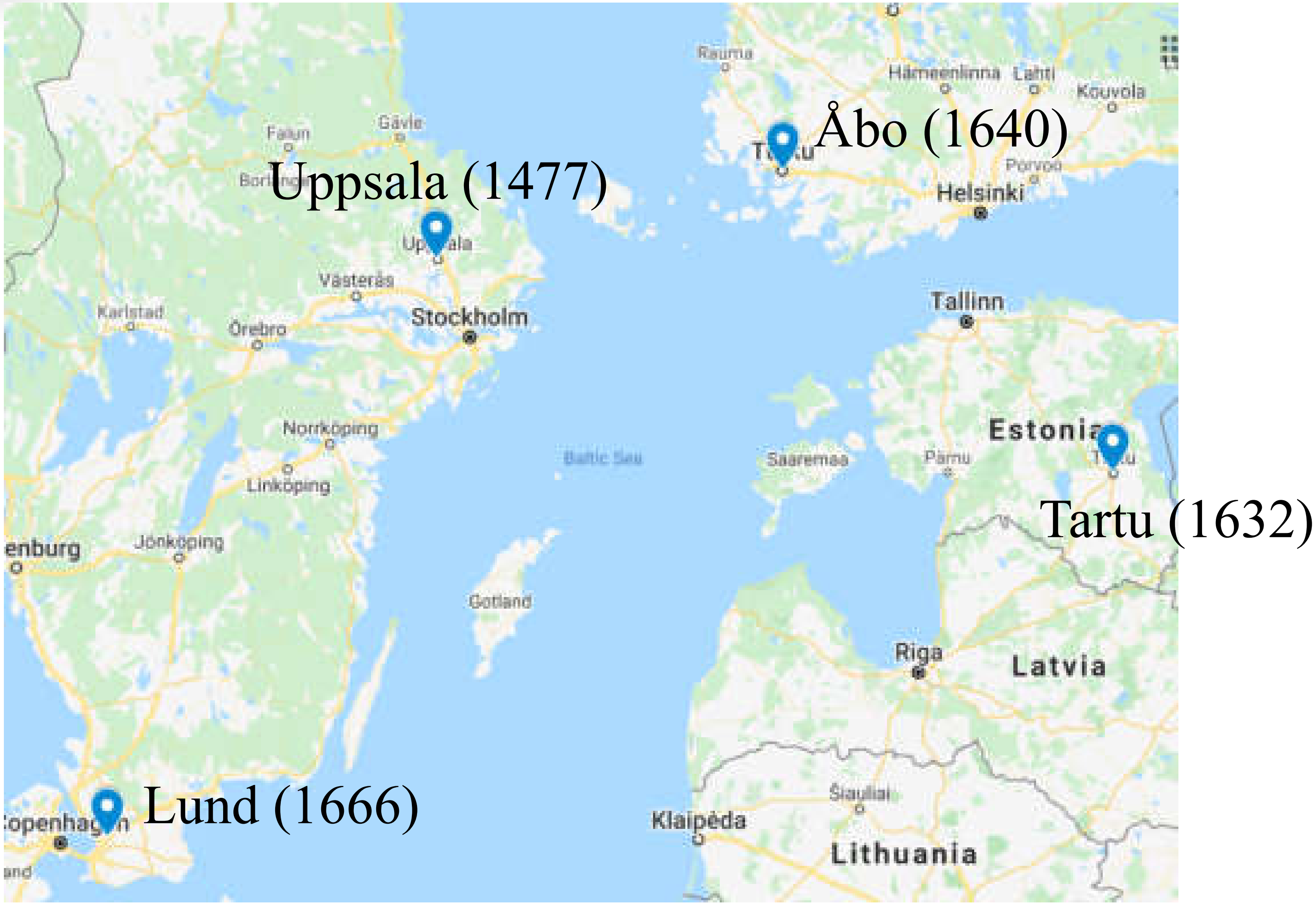
Then linearize the matrix

⇩

1000001001000001

The genotype

The 4 oldest Swedish universities



Genotypes, phenotypes and fitness

and a TSP example

	1	2	3	4
Uppsala (1477)	1	0	0	0
Tartu (now Estonian, 1632)	0	0	1	0
Åbo (now Finnish, 1640)	0	1	0	0
Lund (1666)	0	0	0	1

Then linearize the matrix

⇩

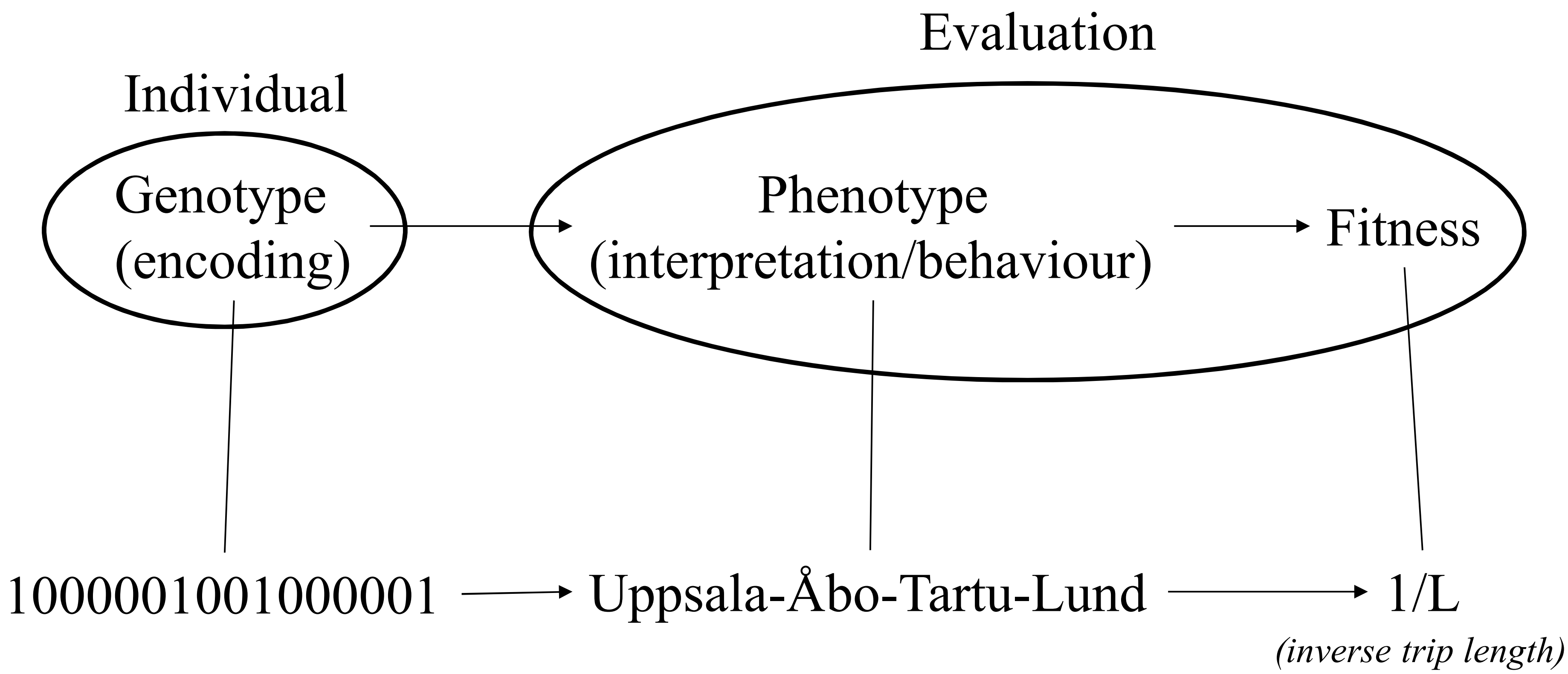
1000001001000001

The genotype

- A genotype's *fitness* (a scalar value) is evaluate by a *fitness function*
  - Requires an interpretation of the genotype
    - *the phenotype*
    - in this example the route Uppsala-Åbo-Tartu-Lund
  - Genotype-phenotype mapping should be many-to-one (but is in practice often many-to-many, also IRL)

Genotypes, phenotypes and fitness

and a TSP example

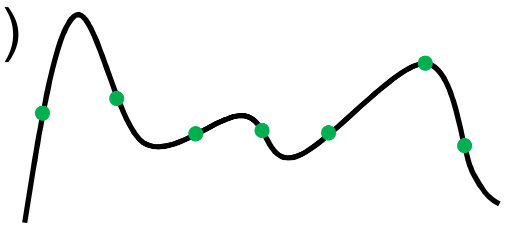


As shown later, though, there are much better ways to represent a route in TSP

Selection

Probabilistic, based on fitness or rank

- Fitness selection: Select individuals with probability proportional to their fitness
  - also known as Roulette wheel, or Bolzmann selection
- Rank selection: First sort the individuals after fitness, then select with a probability proportional to their <u>index</u> in that list, not their fitness
- Rank selection is often better
  - Fitness selection can be unfair if the best individual is much better than the rest OG2
- All individuals should have *some* probability (non-zero) of being selected
  - There is no point having them in the population otherwise
  (and they may contain useful genetic material)
- Allow reselection



Slide 10

OG2 Lägg till något här om MATLABs selection-parameter för rank selection, som blir en sorts mellanting

Olle Gällmo; 2021-03-12

Cloning

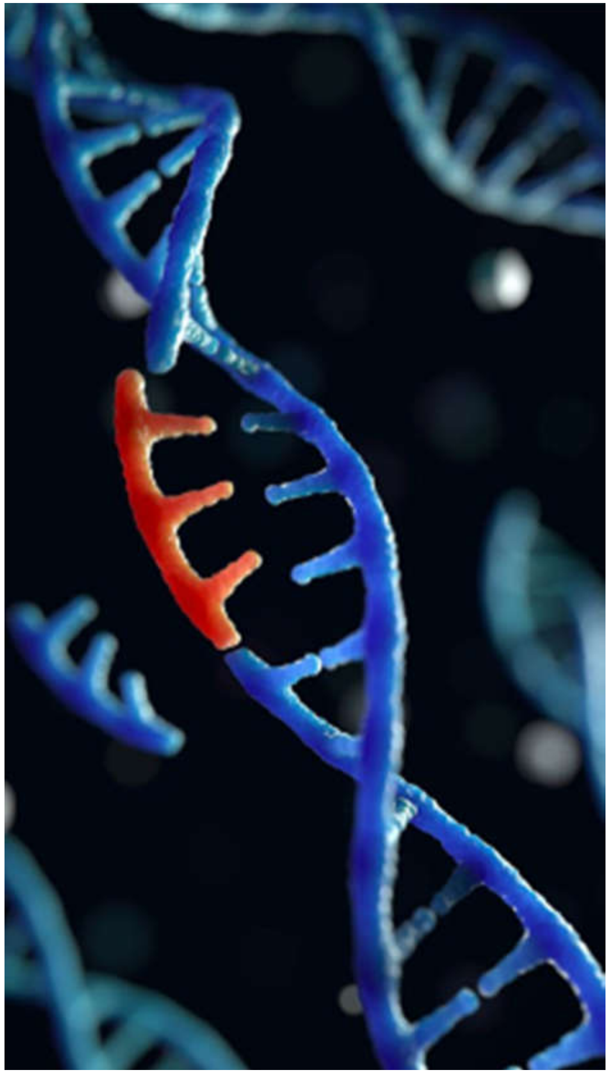
1. Select an individual
    - as above
2. Copy the selected individual, unalterered, to the new population

Dolly, the first mammal clone (1996)

- *Elitism*: Guarantee that the most fit individual(s) are cloned this way
    – to avoid forgetting the best solution(s) found so far
    – but don't overdo it
        - increases risk of premature convergence (introduced later)

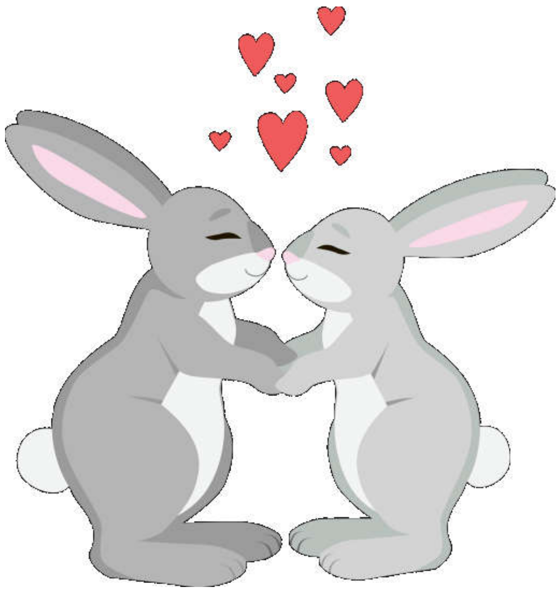
Mutation

- Cloning with noise
- Some part in the copy is altered at random
  - for example by flipping a bit
- Should have very low probability
  - This is not the search operator
  - It's just there to 'shake the box'
- Roughly corresponds to exploration in RL
- It is sometimes suggested to invert selection for this
  - lower fitness ➔ higher probability



Recombination

Crossover



1. Select two individuals
   - as above
2. Perform a *crossover* of the two genotypes, creating two new genotypes/individuals
3. Insert the new individuals in the population

- Crossover (not mutation) is the search operator in (most forms of) evolutionary computing
- If it turns out that new solutions are found mostly through mutation, you should either
  - redesign crossover/genotype to make it work, or
  - be happy that it works anyway, but maybe stop calling it EC

opinion!

UPPSALA UNIVERSITET

Simple crossover examples

for genetic algorithms

One-point crossover

Swap tails

10011|011 ⇕ ⇒ 10011110

10110|110 ⇒ 10110011

Two-point crossover

Swap a sub-section

100|110|11 ⇕ ⇒ 10010111

101|101|10 ⇒ 10111010

Uniform crossover

Select at random from which parent to pick each bit

10011011

? ? ? ? ? ? ? ? ⇒ 10011110

10110110

- Note that this can be very destructive!
- Does it matter?
- Can we make sure that the created individual is valid?

Crossover can be destructive

- There may be constraints to consider
  - which a simple crossover operator doesn't
- In TSP, for example
  - We must visit each city once, but only once
  - We can not be in two cities at the same time
  - With the bit matrix representation used before, each row and column must therefore contain one, but only one, 1
  - If we linearize this and just swap bits, we are very likely to produce invalid solutions

	1	2	3	4
Uppsala	1	0	0	0
Tartu	0	0	1	0
Åbo	0	1	0	0
Lund	0	0	0	1

- Strictly speaking, this is not necessary to prevent
  - Non-functional individuals are unlikely to be selected anyway
  - But it makes crossover inefficient (more like a mutation op)

Better crossover for TSP

- Representations (genotypes) and crossover operators should be designed together!
- If we number the cities, a route could be represented as a list of integers instead
  - For example, for 8 cities: (5 7 1 6 2 4 3 8)
  - It also makes genotype≈phenotype
- Many suggested crossover operators for this repr.
  - For example, Order Crossover (OX)
  - Pick a sub-tour
    - like two-point crossover but we keep what's inbetween the points

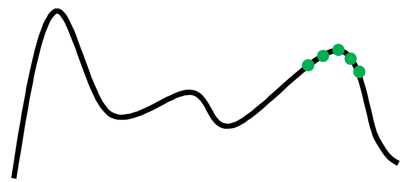
      (5 7 | **1 6 2** | 4 3 8) ==> (4 8 1 6 2 3 7 5)

      (1 7 | **5 4 8** | 2 6 3) ==> (6 2 5 4 8 3 7 1)
  - Fill in the rest by picking cities from the other parent, in order, and excluding already visited ones

UPPSALA UNIVERSITET

Properties of EC
Premature convergence

- EC is parallel search
  - many search points in the search space
- Not much use if they are all in the same place!
- We should avoid *premature convergence*
  - 'convergence' here means that the population has converged to one spot, not necessarily the search
  - that spot may not even be a local optimum



- We want to maintain diversity
  - This is the main task of mutation, injecting new material
- Risk factors: small population sizes, fitness selection (instead of rank selection), too much elitism, poorly designed crossover operators ...

Properties of EC

- EC is unlikely to get stuck in local optima
  - due to the parallel search
    - if we can avoid premature convergence
  - also since EC does not follow gradients
  - mutation helps
    - in theory it can make it impossible to get stuck
- EC does not require the objective function (nor any other function) to be differentiable
  - The objective function (fitness) is used only in <u>selection</u> of individuals, not to decide how to change them
  - We can use the true objective function, not a differentiable approximation (as we must, for example, when training a neural network with Backprop on a classification task)

Properties of EC

- RL is perhaps more clever (more directed) in its search for a solution
  - EC compensates by its more global view
  - as long as we can maintain diversity
- EC (in particular GA) can be used to search for the best combination of parameters which control or define something
  - A neural network is such a parameterized system ...
  - Training neural networks has actually been one of the most common applications of GA (more next lecture)

Applications

- Training neural networks
  - at least shallow ones
- Engineering designs
  - antenna, jet engine and wind power turbines, focusing mirrors ...
- Scheduling
  - schools, trains, airlines, ...
- Bioinformatics
- Game design
  - racing games in particular