Evolution and Genetic Algorithms

Exponential growth and *per capita* growth rate

- R : population size (number of individuals, or, rather, *density*, i.e. indidividuals per area or volume unit)
- *b* : *per capita* birth rate (”how many offspring each individual gets, per time unit”)
- *d* : per capita death rate (”how many times each individual dies, per time unit”)
- Total births per time unit: *bR*
  Total deaths per time unit: *dR*
- Population growth rate: $\frac{\mathrm{d}R}{\mathrm{d}t} = bR - dR = (b - d)R = r\mathrm{R}$
- *r* : intrinsic growth rate
- The *per capita growth rate*, defined as $\frac{\mathrm{d}R}{\mathrm{d}t}$ / R is in this model equal to
  $\frac{\mathrm{d}R}{\mathrm{d}t} / \mathrm{R} = rR / R = r.$
- The *per capita* growth rate is the growth rate per individual. In this model it is given by the model parameter r.
- The equation $\frac{dR}{dt} = r\mathrm{R}$ describes *exponenital growth*.

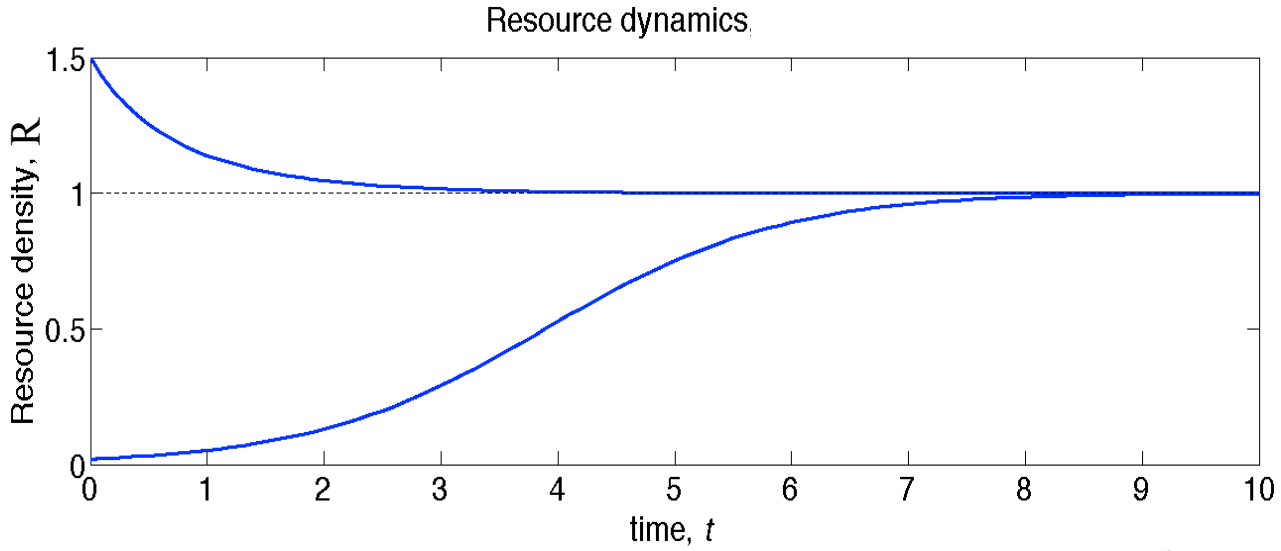
Adding prey density dependence

Resource dynamics:

$$\frac{dR}{dt} = rR\left(\frac{K-R}{K}\right)$$

density dependence

R = resource density
r = intrinsic growth rate
K = carrying capacity



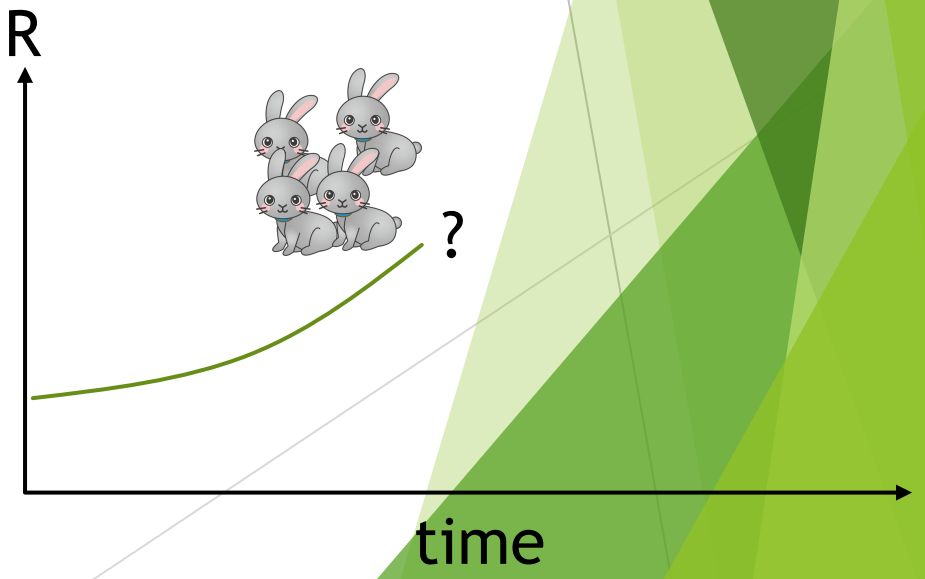
Modelling predation
Prey population growth

Prey population growth = prey births – prey deaths

Prey population growth = prey birth – prey background mortality – predation

$$\frac{dR}{dt} = rR\left(\frac{K-R}{K}\right) - ?$$

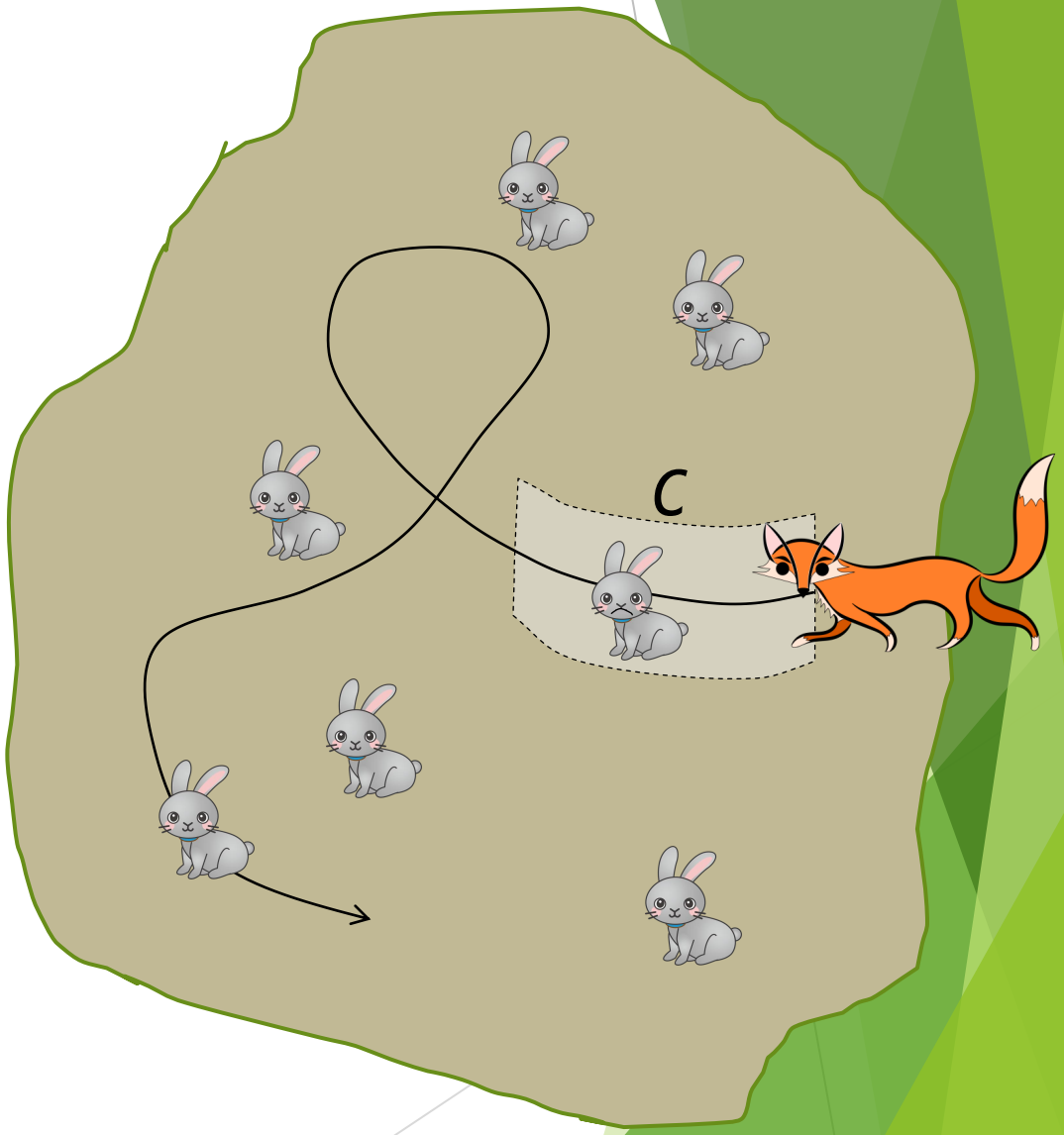
r = prey intrinsic growth rate (b-d)



How many prey does a single predator catch, per time unit?

- Prey density, prey per area unit: *R*
- Search area per predator, per time unit: *a* (predator *attack rate*)
- Prey caught per time unit, per predator: *aR*
- Predator density: C
- Total prey caught: *aRC*
- Total prey growth:

$$\frac{dR}{dt} = rR\left(\frac{K-R}{K}\right) - aRC$$

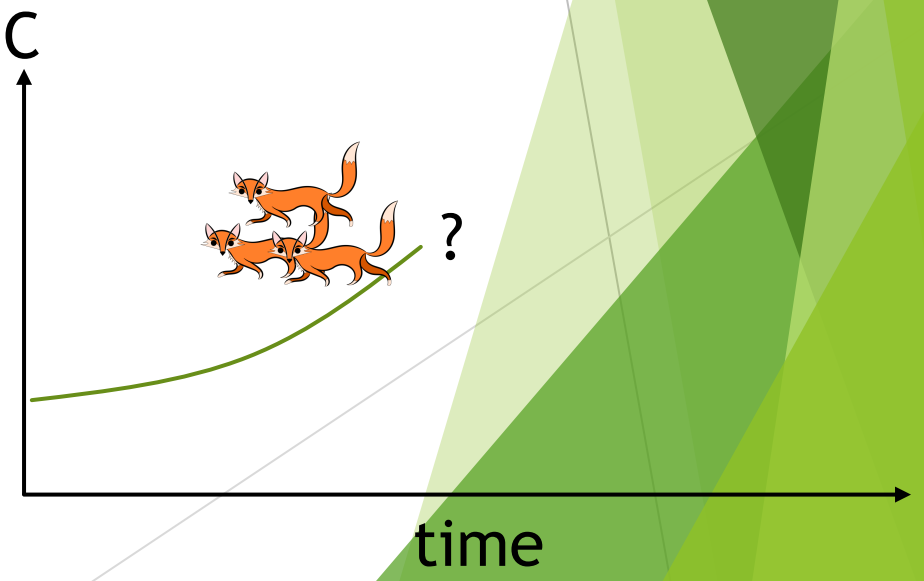


Modelling predation
Predator population growth

Predator population growth =
predator growth due to predation – background predator mortality

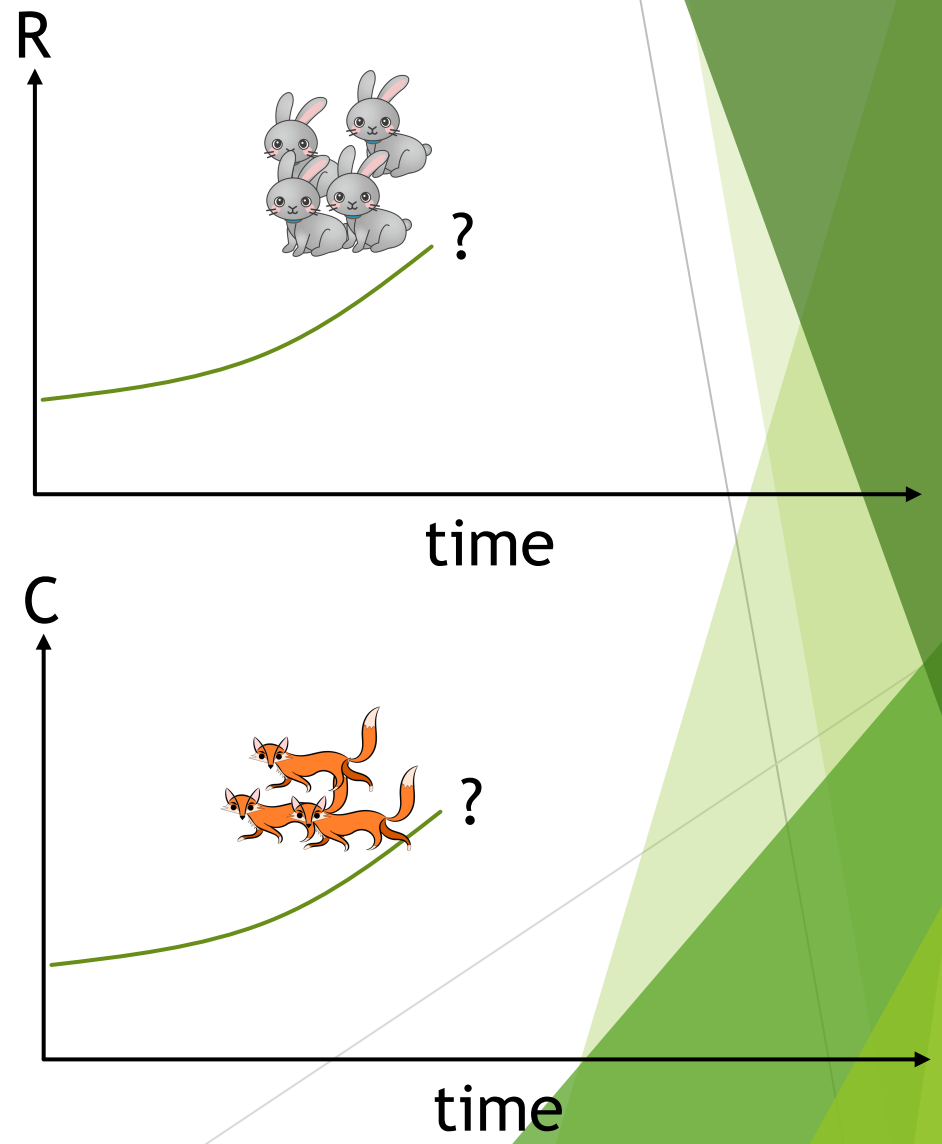
$$\frac{dC}{dt} = \varepsilon a R C - \mu C$$

ε = conversion factor from prey to predators
μ= predator mortality, *per capita*



The complete population dynamics

$$\begin{cases} \dfrac{dR}{dt} = rR\left(\dfrac{K-R}{K}\right) - aRC \\ \dfrac{dC}{dt} = \varepsilon aRC - \mu C \end{cases}$$



The complete population dynamics

R : Resource abundance (e.g. kg/ha)

C: Consumer abundance (e.g. kg/ ha)

a : Consumer attack rate (e.g. ha/year)

ϵ : conversion coefficient (e.g. kg^{-1})

μ : background mortality (e.g. $year^{-1}$)

Fitness = per capita growth rate (why?)

Resource population growth: $\frac{dR}{dt} = fR = rR(1-R)$ - *aRC*

Consumer population growth: $\frac{dC}{dt} = fC = (\epsilon aR - \mu)C$

Resource fitness: $f_R = r(1-R)$ - *aC*

Consumer fitness: $f_C = \epsilon aR - \mu$

Resources consumed per time unit, per consumer.

What if the attack rate a evolves (the comsumers become more efficient)**?**

The complete population dynamics

Resource dynamics:

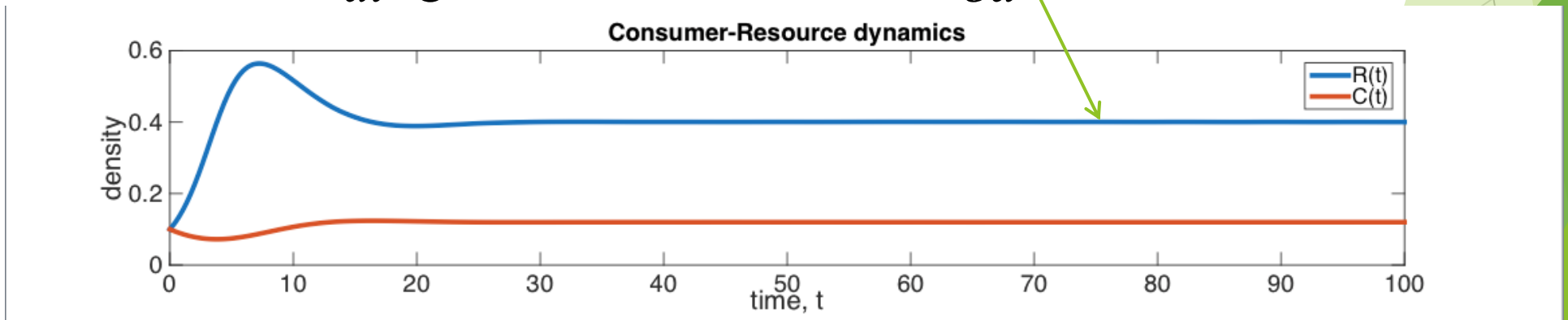
$$\frac{dR}{dt} = rR\left(\frac{K-R}{K}\right) - aRC$$

Consumer dynamics:

$$\frac{dC}{dt} = \varepsilon aRC - \mu C$$

Consumer *per capita* growth (fitness!):

$$\frac{dC}{dt}\frac{1}{C} = \varepsilon aR - \mu \quad \square \quad R^* = \frac{\mu}{\varepsilon a}$$



The complete population dynamics

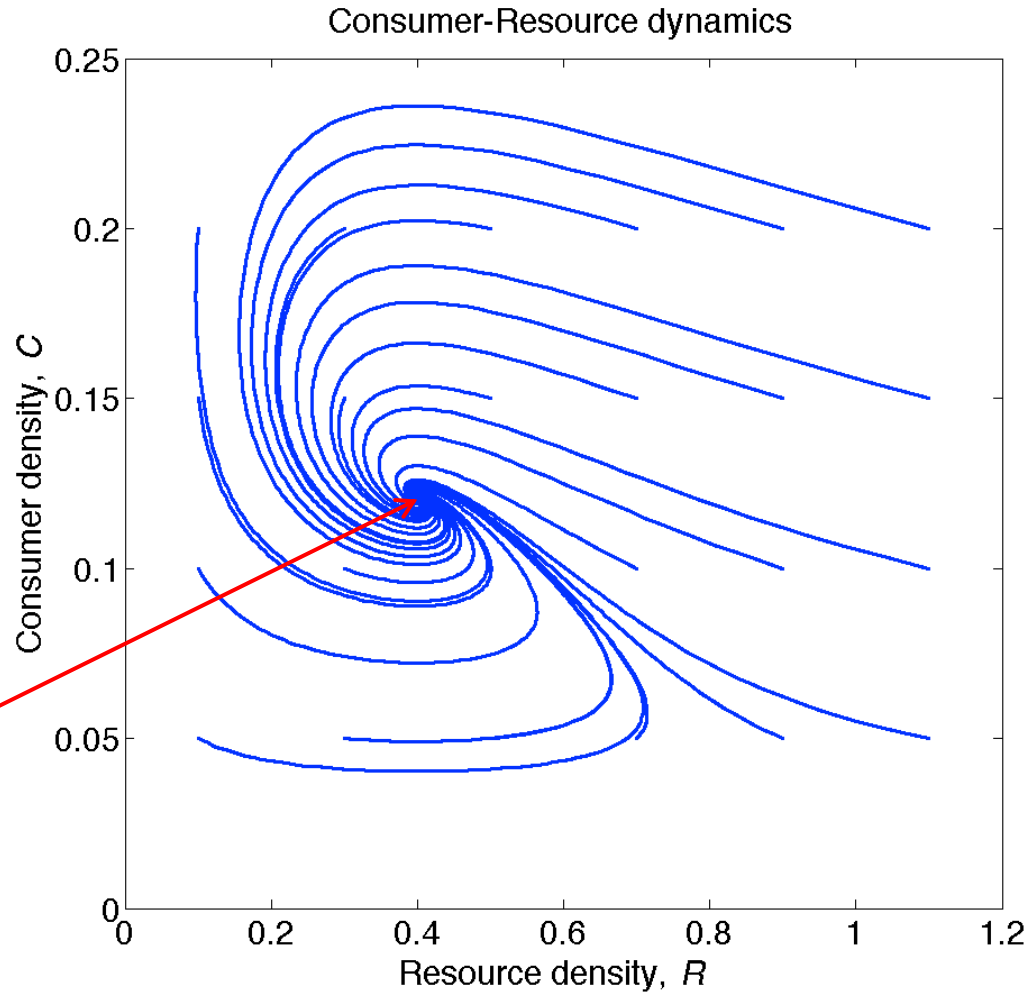
A dynamically stable, fixed point equilibrium

$$\begin{cases} \dfrac{dR}{dt} = rR\left(\dfrac{K-R}{K}\right) - aRC \\ \dfrac{dC}{dt} = \varepsilon aRC - \mu C \end{cases}$$

Substitute C* into the resource equation and solve

Scaling the system by setting *K*=1 (commonly done do simplify the system) leads to

$$(R^*, C^*) = \left(\frac{\mu}{\varepsilon a}, \frac{r}{a}\left(1 - \frac{\mu}{\varepsilon a}\right)\right)$$



What if the attack rate a evolves (the comsumers become more efficient)?

Consumer fitness: $f_C = \epsilon a R - \mu$

First, sort out the ecological dynamics (we already did).

Second, we need a theoretical evolutionary foundation (e.g. Game Theory)

Third, we need a evolutionary theory and eco-evolutionary model analyses (e.g. Adaptive Dynamics)

Evolving attack rate

Assume the consumer attack rate a depends on some heritable trait x.

In other words, assume there can be several types of consumers present in the population, each with its own trait value x and attack rate $a(x)$:

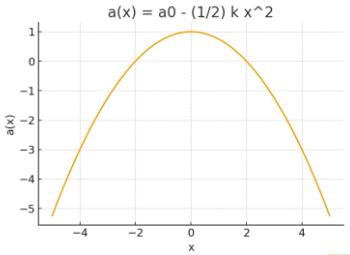
A consumer with trait x thus has *per capita growth rate (fitness)*

Trait-based fitness

$$f(x) = \epsilon a(x) R - \mu$$

Trait-based attack rate

$$a(x) = a_0 - \frac{1}{2} k x^2$$



The trait-based resource dynamics becomes:

$$\frac{dR}{dt} = rR(1-R) - \sum_{i=1}^{C} a(x_i) R$$

C = the number of consumers

Adaptive Dynamics assumptions

Assume mutations are rare, such that:
1. There is most of the time only one type (the resident) present in the population

2. An ecological equilibrium is established before any new mutant occurs

If a resident type *x* has established an ecological equilibrium

$$(R^*, C^*) = \left(\frac{\mu}{\varepsilon a(x)}, \frac{r}{a(x)}\left(1 - \frac{\mu}{\varepsilon a(x)}\right) \right),$$

what is then fitness of a rare invading mutant type x'?

$$f(x', x) = \varepsilon a(x') R^*(x) - \mu = \mu \left(\frac{a(x')}{a(x)} - 1 \right)$$

This is the *invasion fitness*.

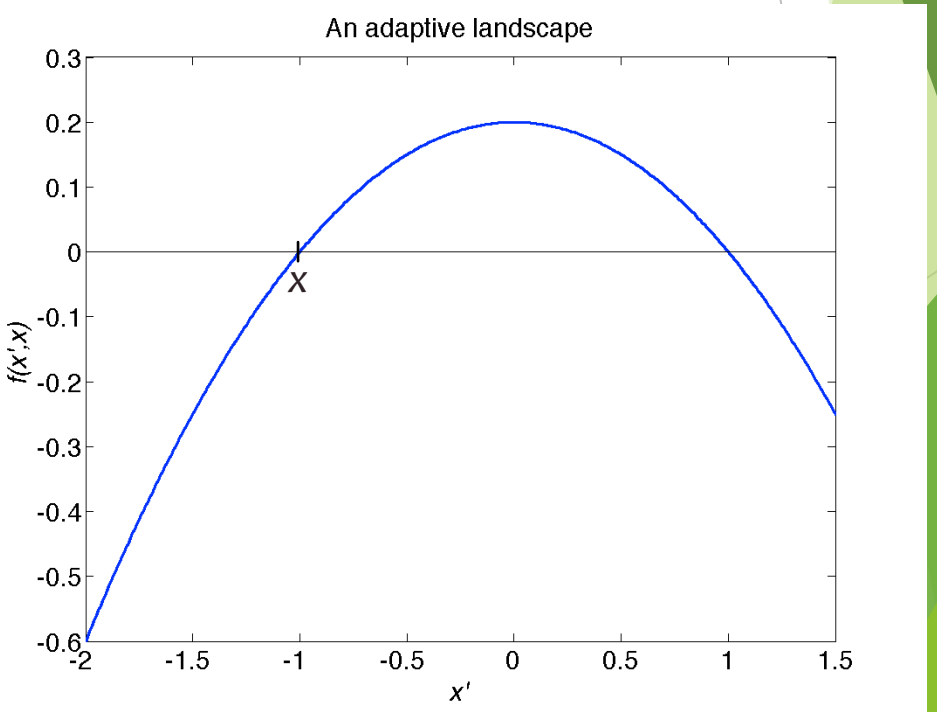
Invasion fitness

General definition:

$f(x',x)$ = per capita growth rate of rare type x' in the equilibrium environment set by resident type x.

By assumption: $f(x,x) = 0$

Keeping resident type x fixed, but varying the mutant type x' generates an *adaptive landscape*.



Trait substitution sequence

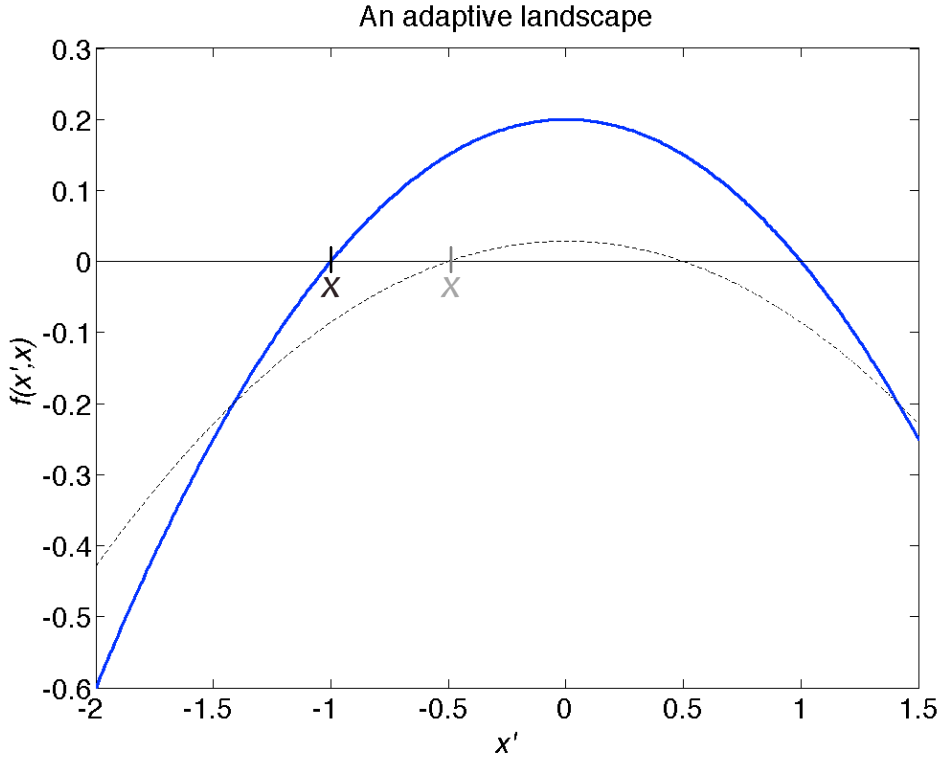
A mutant with positive invasion fitness can invade and replace the resident type, establishing a new equilibrium.

Evolution will in this way proceed in a ‘trait substitution sequence’.

The adaptive landscape shows which mutants can invade, but the landscape will change when the resident trait changes.

Always, $f(x,x) = 0$.

Evolution is in this way ‘climbing a sinking landscape’.



Genotype–Phenotype Mapping in Adaptive Models

Overview

- Multilocus diploid genotypes dictate traits
- Multiple additive alleles effects traits
- Traits of particular interest: ecological and mating traits
- Adaptation emerges from genetic variation, GP mapping and selection

Example: Genotype Architecture in evolutionary models

- Each evolving trait is encoded by:
  - Multiple diploid loci (20–60 per trait)
  - Diallelic alleles, each with some numeric value
  - Additive allele effects (sum across loci)
  - After allele summation, traits are rescaled into continuous trait values
  - Recombination leads to new variants on which selection can act
  - Mutations lead to new variations on which selection can act

How Phenotypes Determine Fitness & Mating

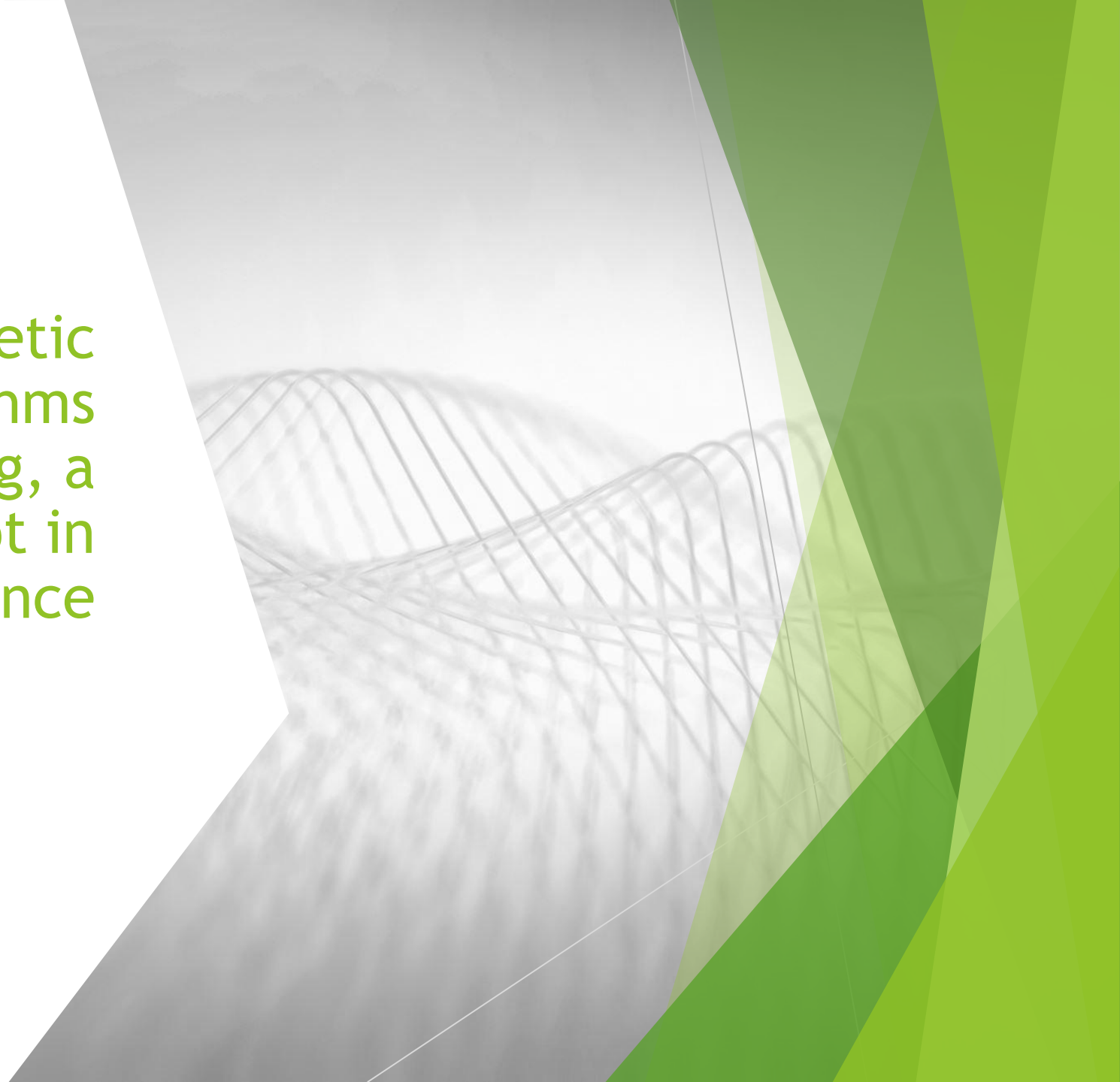
- Phenotype values affect:
  - Resource uptake
  - Habitat survival
  - Assortative mating via trait similarity
  - Sexual selection

How Adaptation Emerges

- Adaptive dynamics in the model arise from:
  - Ecological selection traits
  - Sexual & assortative mating selection on traits
  - Mutation introducing variation
  - Recombination mixing genotypes

Evolution and Genetic Algorithms

- Evolutionary computing, a general concept in computational science



Evolutionary computing

- Algorithmic models that use Darwinian-like evolution
- Iterative optimization
- E.g. Evolutionary strategy models
- E.g. Genetic algorithms
- E.g. Genetic programming

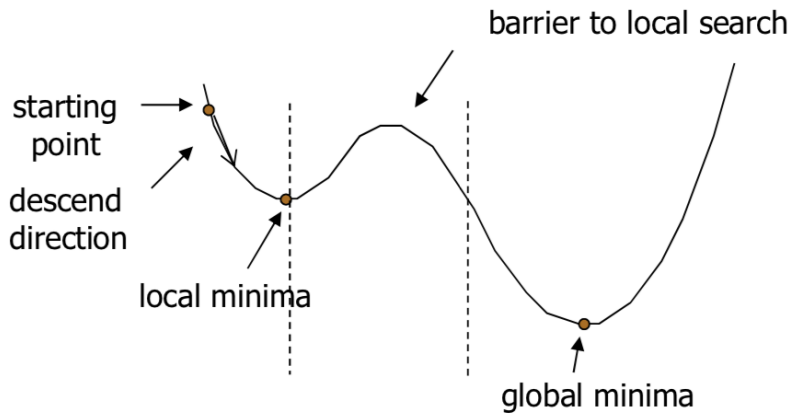
Evolutionary computing

- Algorithmic models that use Darwinian-like evolution
- Iterative optimization
- E.g. Evolutionary strategy models
- E.g. Genetic algorithms
- E.g. Genetic programming

- Key components:
  - Selection (better solutions survive)
  - Mutation (random change)
  - Crossover/ recombination (combining parts)
  - Inheritance (solutions pass on structure to next iteration)

Genetic algorithms, GA

- Robust and global compared to calculus and enumerative solutions
- Easier to use when little is known about the problem
- Can search a large solution space and find a global optimum
- Use only for complex problem (expensive)



GA, step by step

1. Represent your problem as genes in a binary value chromosome (= solution)

Example:

Find the maximum of a 10 digits base 2 number:

```
MyChromosome <- c(1, 0, 0, 0, 1, 0, 1, 1, 0, 0 )
```

Maximum Value of a 10-Digit Binary Number, known solution

- A 10-digit base-2 number has 10 bits, each being either 0 or 1.
  - The largest value is achieved when all bits are set to 1: 1111111111_2
  - Each bit represents a power of 2: 2^9, 2^8, ..., 2^1, 2^0
  - Therefore: $1111111111_2 = 2^9 + 2^8 + \ldots + 2^1 + 2^0$
  - This is a geometric series: sum = $2^{10} - 1 = 1023$
  - A 10-bit number can represent a total of $2^{10} = 1024$ distinct values (0 to 1023).

GA, step by step

2. Random generation of a ”population” of chromosomes

individual	representation
1	0 0 1 0 1 1 1 0 1 0
2	1 0 1 1 1 0 1 0 1 0
3	0 0 0 1 1 1 1 0 0 1
4	1 0 0 0 0 1 1 1 1 0

In reality start with 50 chromosomes (=solutions)

GA, step by step

3. Evaluate ”fitness” of individual chromosomes

individual	representation	fitness
1	0 0 1 0 1 1 1 0 1 0	186
2	1 0 1 1 1 0 1 0 1 0	746
3	0 0 0 1 1 1 1 0 0 1	121
4	1 0 0 0 0 1 1 1 1 0	532

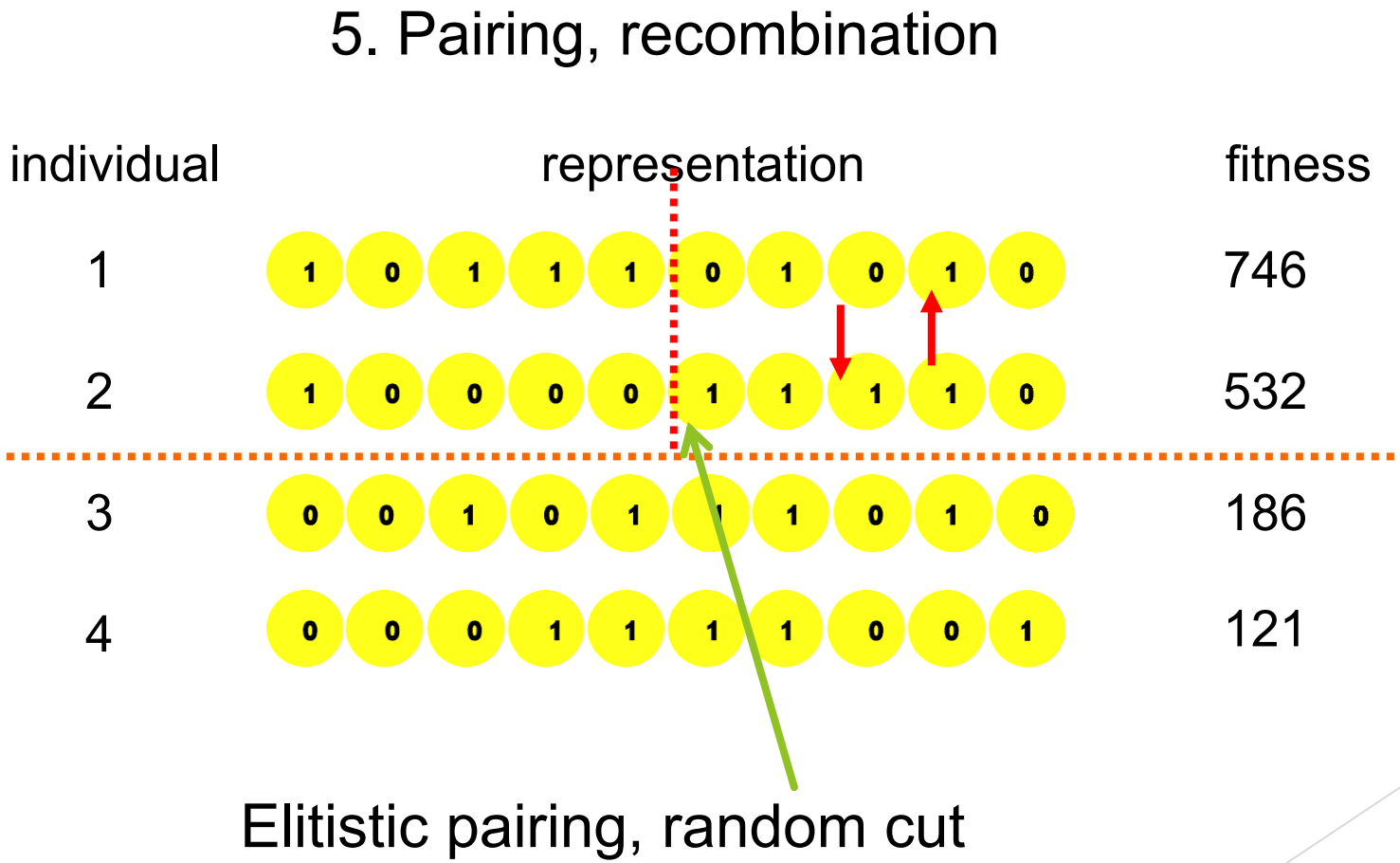
Fitness = how good is a chromosome as solution

GA, step by step

4. Sort solutions after rank

individual	representation	fitness
1	1 0 1 1 1 0 1 0 1 0	746
2	1 0 0 0 0 1 1 1 1 0	532
3	0 0 1 0 1 1 1 0 1 0	186
4	0 0 0 1 1 1 1 0 0 1	121

GA, step by step



GA, step by step

5. Pairing, recombination → offspring

individual	representation	fitness
1	1 0 1 1 1 0 1 0 1 0	746
2	1 0 0 0 0 1 1 1 1 0	532
3	0 0 1 0 1 1 1 0 1 0	186
4	0 0 0 1 1 1 1 0 0 1	121
5	1 0 1 1 1 1 1 1 1 0	767

High quality offspring = good solution

GA, step by step

6. Insert offspring in population, discard worst solutions

individual	representation	fitness
1	1 0 1 1 1 1 1 1 1 0	767
2	1 0 1 1 1 0 1 0 1 0	746
3	1 0 0 0 0 1 1 1 1 0	532
4	0 0 1 0 1 1 1 0 1 0	186
5	0 0 0 1 1 1 1 0 0 1	121

”killed”! (keep the number of chromosmes at 50)

GA, step by step

Why cannot the GA solve this problem?

individual	representation	fitness
1	1 0 1 1 1 1 1 1 1 0	767
2	1 0 1 1 1 0 1 0 1 0	746
3	1 0 0 0 0 1 1 1 1 0	532
4	0 0 1 0 1 1 1 0 1 0	186

GA, step by step

7. Mutation

individual	representation	fitness
1	1 0 1 1 1 1 1 1 1 0	767
2	1 1 1 1 1 0 1 0 1 0	874
3	1 0 0 0 0 1 1 1 1 0	532
4	0 0 1 0 1 1 1 0 1 0	186

GA, step by step

7. Mutation

individual	representation	fitness
1	1 0 1 1 1 1 1 1 1 0	767
2	1 1 1 1 1 0 1 0 1 0	874
3	1 0 0 0 0 1 1 1 1 0	532
4	0 0 1 0 1 1 1 0 1 0	186

Now you can make a genetic algorithm!

GA, step by step

8. Cycle until terminantion

1. Represent your problem as genes in a chromosome
2. Random generation of a population of chromosomes
3. Evaluate fitness of individual chromosomes
4. Sort after rank
5. Pairing, recombination
6. Insert offspring
7. Mutation

GA, step by step

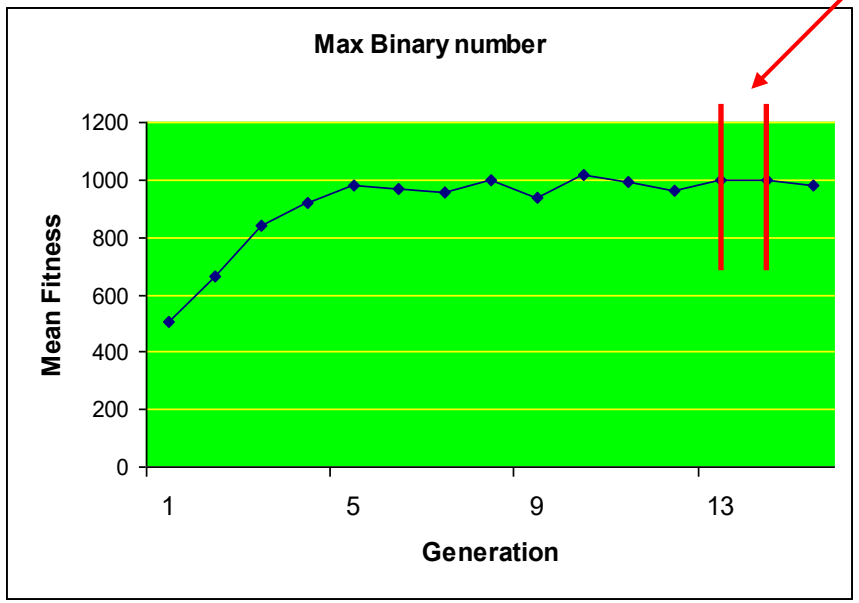
8. Cycle until terminantion

1. Represent your problem as genes in a chromosome
2. Random generation of a population of chromosomes
3. Evaluate fitness of individual chromosomes
4. Sort after rank
5. Pairing, recombination
6. Insert offspring
7. Mutation

Termination?

8. Cycle until terminantion

diff = $y_1 - y_2$

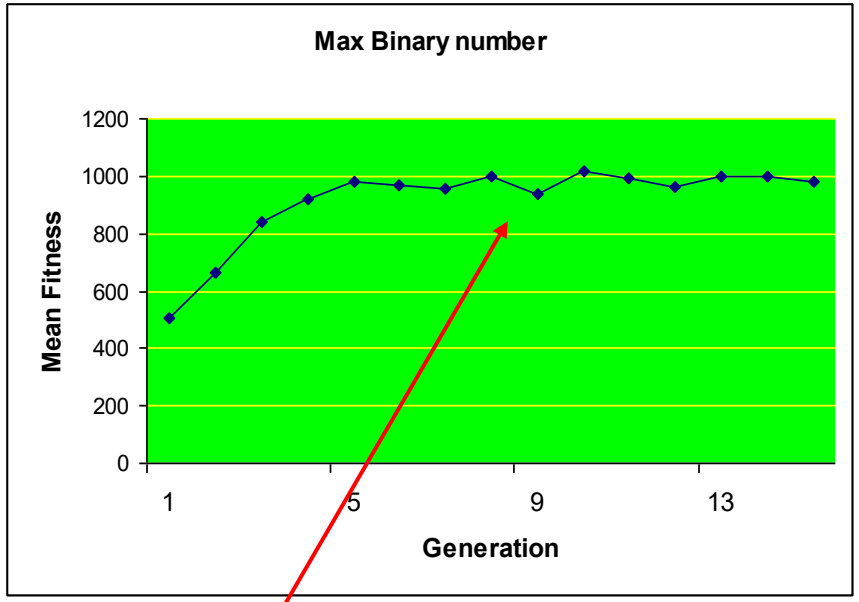


for (g in 1:15)
{cycle algorithm}

g = generation

while (diff > 10)
{diff = $y_1 - y_2$;
cycle algorithm
}

8. Cycle until terminantion



Why not a straight line? It is a very simple problem.

Genetic algorithm

Example 1, binary (= discreet) GA:

Find maximum of a 10 digits base 2 number:

```
MyChromosome <- c(1, 0, 0, 0, 1, 0, 1, 1, 0, 1)
```

Example 2, find best solution of an equation with two unknowns

$f(x, y) = x \sin(4x) + y \sin(2y)$

```
MyChromosome <- c( 3.43537, -1.0002345)
```

Alternative Ways to Solve f(x, y) Optimization

- Equation: $f(x, y) = x \cdot \sin(4x) + y \cdot \sin(2y)$
  - Gradient-based optimization (e.g., BFGS, Newton, Conjugate Gradient)
  - Solve analytically via partial derivatives: $\partial f/\partial x = 0$ and $\partial f/\partial y = 0$
  - Multi-start local optimization to avoid local minima
  - Global optimization alternatives: simulated annealing, basin hopping
  - For bounded domains: grid search or fine 2D sampling

Continuous genetic algorithms:

- Real values can be used instead of binary strings

- Decide range for gene values (e.g. $-5 \leq x \leq 5$)

Best solution of equation with two unknowns:

$f(x, y) = x \sin(4x) + y \sin(2y)$

Individual values (start with randomly created numbers)

1	[3.435, -1.003]
2	[-2.331, - 4.234]
3	[-0.877, - 4.954]
4	[3.537, 1.645]
5	[2.479, - 1.541]

Continuous genetic algorithms:

- problems with recombination:

ind	values
1	[3.435, -1.003]
2	[-2.331, - 4.234]
3	[-0.877, - 4.954]
4	[3.537, 1.645]
5	[2.479, - 1.541]

Question1: Why is direct crossover not good?

Continuous genetic algorithms:

- problems with recombination:

ind	values
1	[3.435, -1.003]
2	[-2.331, - 4.234]
3	[-0.877, - 4.954]
4	[3.537, 1.645]
5	[2.479, - 1.541]

- Key problems with direct crossover:
  - Breaks good (x, y) combinations
  - Creates poor-quality offspring
  - Does not fit continuous-valued search spaces
  - Offspring may fall in bad regions
  - Better operators exist (arithmetic, BLX-α, SBX)

Question1: Why is direct crossover not good?

Continuous genetic algorithms:

- problems with recombination:

ind	values
1	[3.435, -1.003]
2	[-2.331, - 4.234]
3	[-0.877, - 4.954]
4	[3.537, 1.645]
5	[2.479, - 1.541]

Question 1: Why is direct crossover not good?
Question 2: Why is averaging better but not good?

Continuous genetic algorithms:

- problems with recombination:

ind	values
1	[3.435, -1.003]
2	[-2.331, - 4.234]
3	[-0.877, - 4.954]
4	[3.537, 1.645]
5	[2.479, - 1.541]

Averaging (arithmetic crossover) improves over direct crossover, because:

- Offspring stay between parents (more stable than direct mixing)
- Less disruption of good (x, y) combinations
- Reduces the chance of creating extreme or invalid values

But it is still not ideal because:

- Averaging reduces diversity too quickly
- Cannot explore beyond parent region (no global exploration)
- May get stuck in local optima

Question 1: Why is direct crossover not good?
Question 2: Why is averaging better but not good?

Continuous genetic algorithms:

- Blending (recombination operator in continuous GA)

ind	values
p_1	[3.435, -1.003]
p_2	[-2.331, - 4.234]

Three solutions:

i) proportional blending: $b * p_1 + (1 - b) * p_2$ where $0 \geq b \geq 1$

ii) linear blending gives three offspring:

offsp1 = $0.5p_1 + 0.5p_2$

offsp2 = $1.5p_1 - 0.5p_2$

offsp3 = $1.5p_2 - 0.5p_1$

iii) combination between crossover and blending

Continuous genetic algorithms:

Question: What problem may this blending process create?
(Think about the min and max values -5, 5)

Suggest a solution to this problem

Continuous genetic algorithms:

- Individuals can be ”out of range”

offsp2 = $1.5p_1 - 0.5p_2$

p1 = 4.952, p2 = -4.213 gives offspring 9.535

Three solutions:

i) chop at limit

ii) ”gene repair”

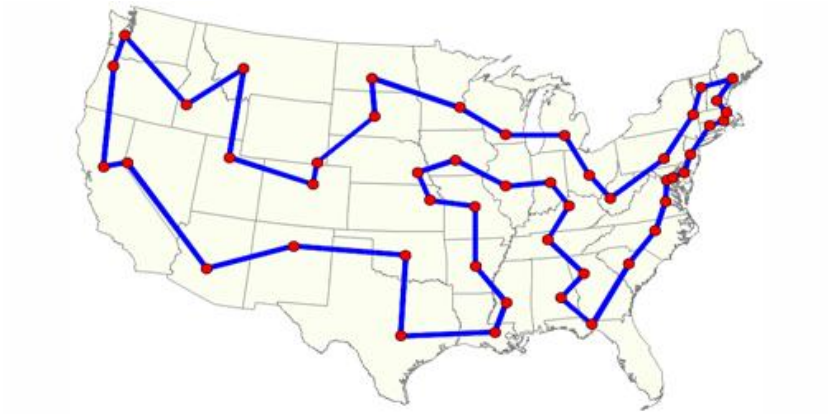
iii) give low fitness

Continuous vs discrete genetic algorithms:

Discrete (0 1 1 0 1)	Continuous (0.654, 3.564…)
binary	real numbers
fitness function	direct use of numbers
simple crossing over	complex mating
traditional, original	newer

Traveling salesman problem:

You have to visit a number of geographic positions, calculate the shortest route



Traveling salesman problem:

You have to visit: Göteborg, Borås, Norrköping, Kristianstad, start in Lund

Traveling salesman problem:

You have to visit: Göteborg, Borås, Norrköping, Kristianstad, start in Lund

Chromosome 1	Lund	Göteborg	Borås	N-köping	K-stad
Chromosome 2	Lund	Göteborg	N-köping	K-stad	Borås
Chromosome 3	Lund	K-stad	Göteborg	Borås	N-köping

Etc.

Traveling salesman problem:

You have to visit: Göteborg, Borås, Norrköping, Kristianstad, start in Lund

Chromosome 1	Lund	Göteborg	Borås	N-köping	K-stad
Chromosome 2	Lund	Göteborg	N-köping	K-stad	Borås
Chromosome 3	Lund	K-stad	Göteborg	Borås	N-köping

Etc.

What is fitness?

Traveling salesman problem:

You have to visit: Göteborg, Borås, Norrköping, Kristianstad, start in Lund

	Lund	Göteborg	Borås	N-köping	K-stad
Lund	0	262	275	431	77
Göteborg	262	0	63	311	264
Borås	275	63	0	250	264
N-köping	431	311	250	0	392
K-stad	77	264	264	392	0

Traveling salesman problem:

You have to visit: Göteborg, Borås, Norrköping, Kristianstad, start in Lund

- TSP chromosome and fitness evaluation:
  - Chromosome = a permutation of cities (route order)
  - Example: [1, 5, 2, 3, 4] means visiting cities in that sequence
  - Distance matrix provides distance between each city pair
  - Total route distance = sum of distances between consecutive cities
  - Return to start city is included in route length
  - Fitness = 1 / total distance (shorter route → higher fitness)

Traveling salesman problem:

You have to visit: Göteborg, Borås, Norrköping, Kristianstad, start in Lund

- **GA works well for the Traveling Salesman Problem:**
  - TSP search space grows factorially (n! routes), too large for brute force
  - GA can efficiently explore huge combinatorial spaces
  - Works well without problem-specific assumptions
  - Maintains a population of routes → avoids getting stuck in bad local optima
  - Specialized crossover/mutation operators preserve valid permutations
  - Produces good near-optimal solutions quickly, even when exact solution is hard
  - Robust against noisy or changing distance data