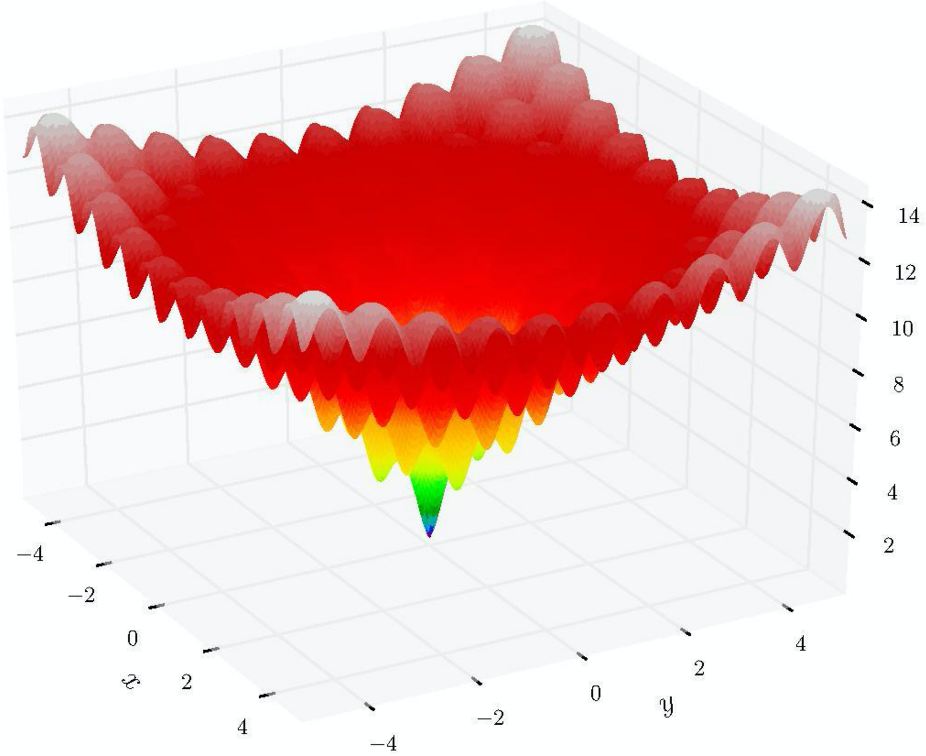
7 Representation, Mutation and Recombination Part 2: Real Number Representations

Real-Valued (Floating-Point) Representation

- many problems occur as real valued problems
- with real parameter values - such as length or height- leading to real fitness values

$$f : \mathbb{R}^n \rightarrow \mathbb{R}$$

- for example Ackley’s function
- which is often used to test EC



$$f(x) = -20 \cdot \exp\left(-0.2\sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} x_i^2}\right)$$

$$-\exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e$$

A Compromise

- in a computer we represent real values using floating point notation
- but all floating point representations have minimum and maximum values
- and a limit of precision
- so we need to be aware that we cannot represent all possible solutions using floating point representation
- we need to make a compromise between range and precision
- but for mutation and recombination we treat floating point values as if they come from a continuous distribution
- meaning that the forms of mutation used for integer reps do not apply

Real-valued Representation: Uniform Mutation

- a genotype is a set of n floating point values:

  $\langle x_1, \ldots, x_n \rangle$

- each value x_i has upper and lower bounds L_i and U_i
- the values can be mutated to a new set of values:

  $\langle x_1, \ldots, x_n \rangle \rightarrow \langle x'_1, \ldots, x'_n \rangle$

- uniform mutation:
  - the x'_i are drawn randomly from $[L_i, U_i]$
  - analogous to bit-flipping (binary) or resetting (integers)
  - so usually each gene has same probability of mutation p_m

Real-valued Representation: Non-uniform Mutation

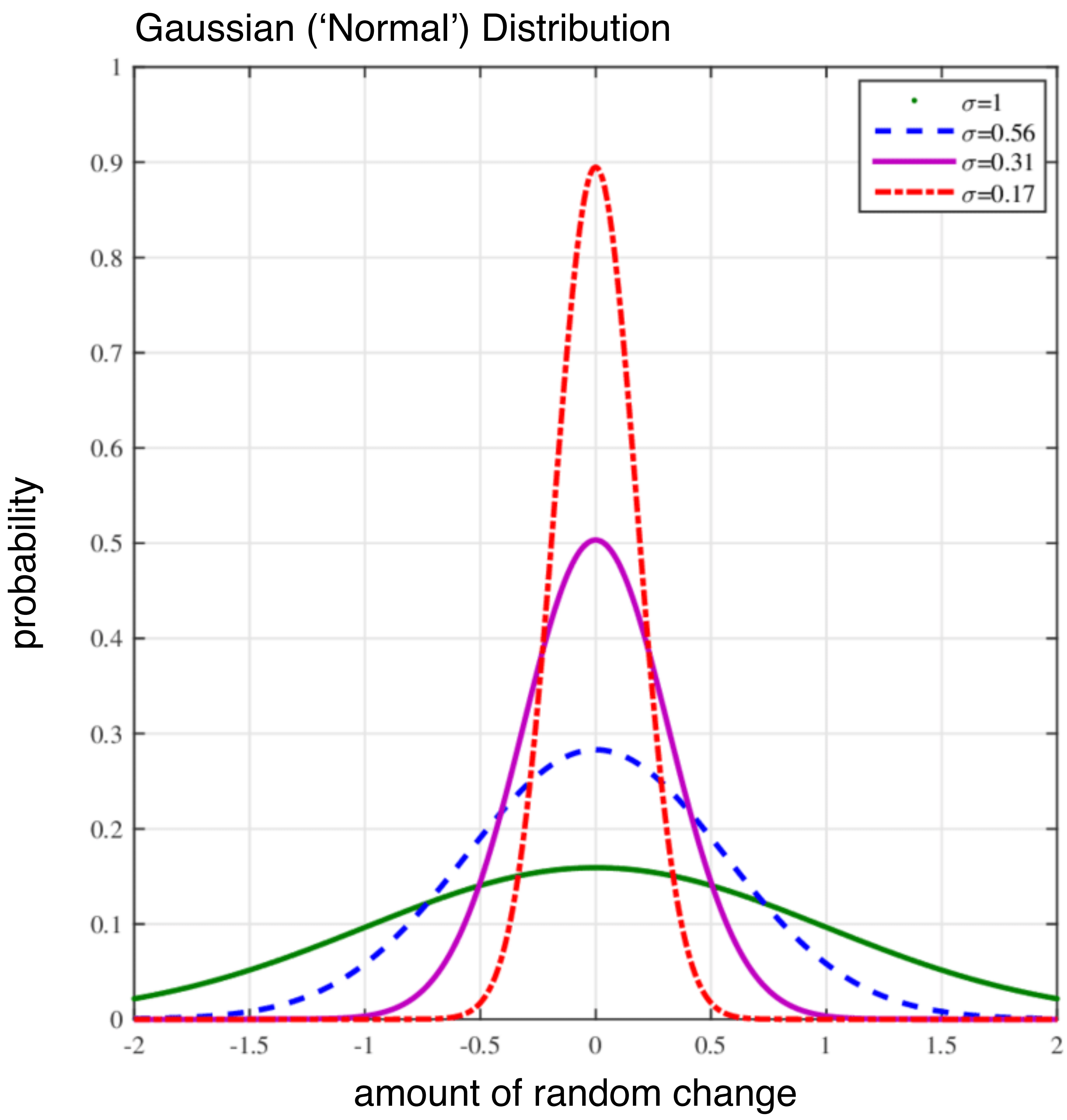
- most common form of mutation for real-valued representation
- analogous to creep method for integers
- usually the amount of change introduced is small
- most common method is to add a random amount to each variable separately, taken from Gaussian distribution $N(0,\sigma)$
    - with mean 0 and standard deviation σ

      $x'_i = x_i + N(0, \sigma)$

        - ensuring that the new value x'_i lies within bounds L_i and U_i
- most of the changes will be very small
- but there is a non-zero probability of generating very large changes

Real-valued Representation: Non-uniform Mutation

- the amount of change to be applied comes from this distribution
- σ determines the extent of the change
- σ is the mutation step size
- the probability distribution curve never actually reaches zero
- but at some point it falls below the level of precision of the floating point representation



Real-valued Representation: Self-Adaptive Mutation

- here, step-sizes are included in the genotype and undergo variation and selection themselves:

  $\langle x_1, \dots, x_n, \sigma \rangle$

- the mutation step size is not set by user but coevolves with solution
- why do this?
- because different mutation strategies may be appropriate in different stages of the evolutionary search process
- we can evolve a gene for one σ that applies to all x (as above), or we can evolve n separate σ_i for each x_i:

  $\langle x_1, \dots, x_n, \sigma_i, \dots, \sigma_n \rangle$

Real-valued Representation: Self-Adaptive Mutation

- order of play:
  - mutate σ first: $\sigma \rightarrow \sigma'$
  - *then* mutate the rest of the genotype: $x_i \rightarrow x' + N(0,\sigma)$
  - net mutation effect:

    $$<x_1, \ldots, x_n, \sigma> \rightarrow <x'_1, \ldots, x'_n, \sigma'>$$
- why does it have to be in this order?
  - if we mutate σ last then during selection its effect won't be evaluated, but the value of the previous σ will be instead
- so the new genotype is evaluated twice:
  - primary: x' is good if $f(x')$ is good
  - secondary: σ' is good if the x' it created is good

Real-valued Representation: Uncorrelated mutation with one σ value

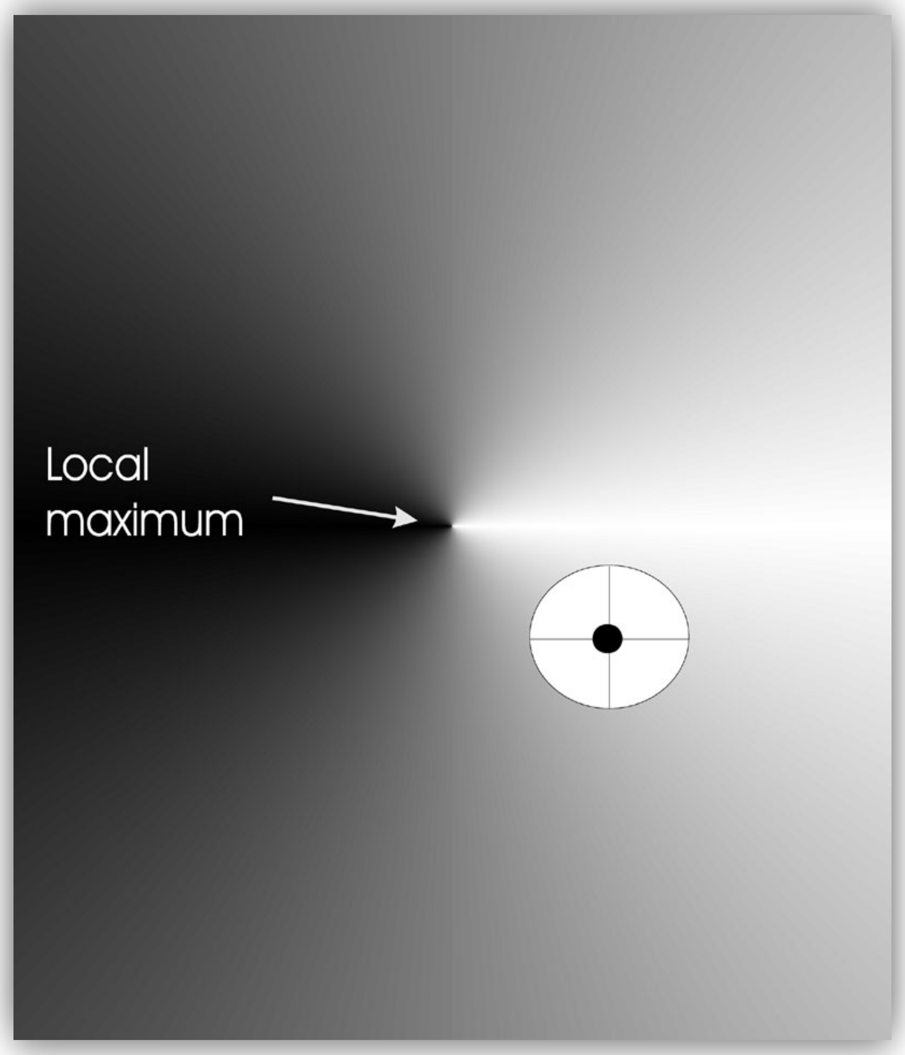
- genotype has a single value for σ that determines the shape of the distribution that will be used to mutate each x_i:

  $<x_1, \ldots, x_n, \sigma>$

- σ is mutated by multiplying it by an amount whose calculation contains two key terms:
  - a random value from N(0,1)
  - a learning rate parameter τ that can be set by the user
- typically τ is proportional to $1/\sqrt{n}$ which ensures:
  - smaller modifications occur more often than large ones
  - equal probability of σ increasing or decreasing
- because too-small a step size would have negligible effect, we usually set a lower bound ε for σ:

  $\sigma < \varepsilon \Rightarrow \sigma = \varepsilon$

Real-valued Representation:
Uncorrelated mutation with one σ value



- we are looking at a fitness landscape from above
- the dot represents an individual
- the perimeter of the circle represents mutants with the same chance of being created from that individual

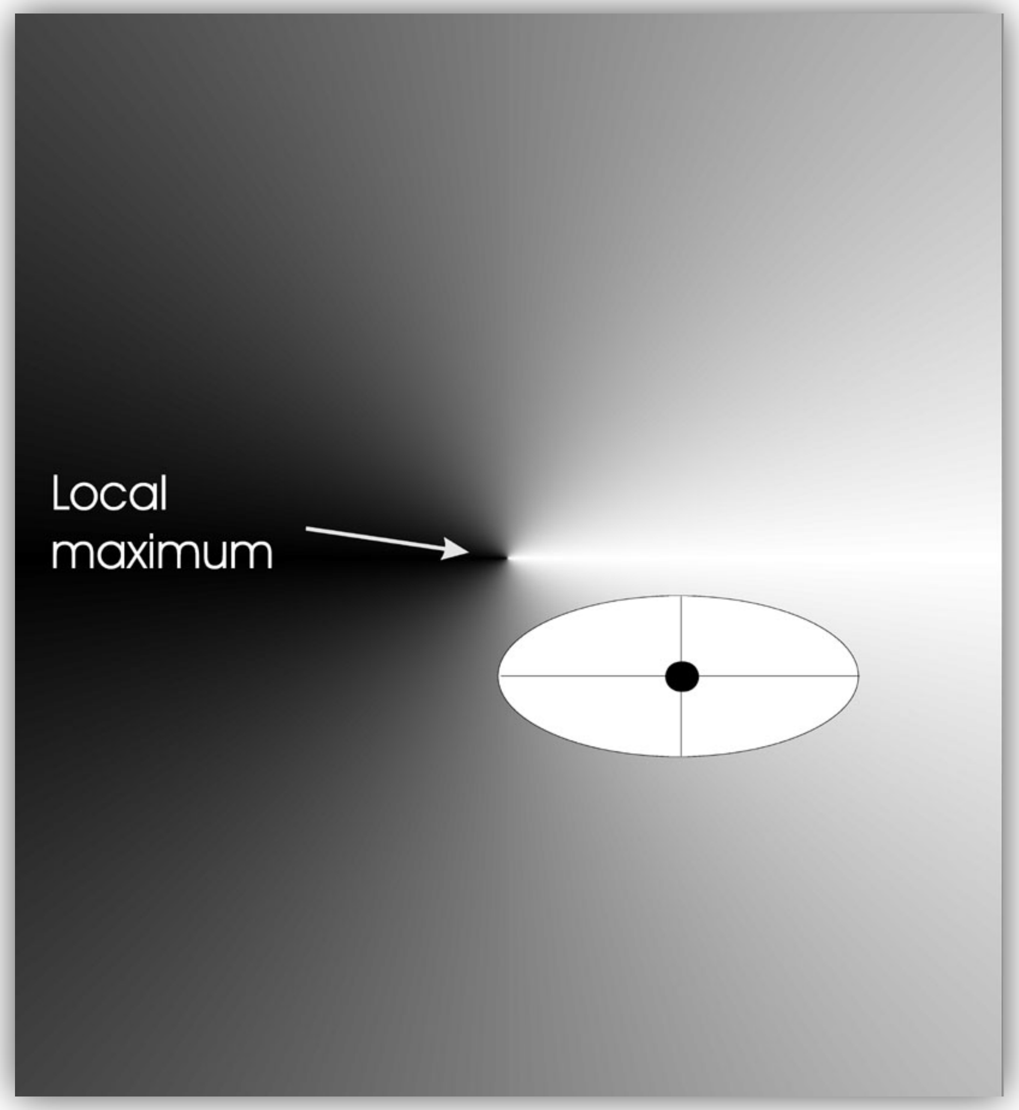
Real-valued Representation: Uncorrelated mutation with n σ values

- genotype has n separate σ_i for each x_i:

  $<x_1, \ldots, x_n, \sigma_i, \ldots, \sigma_n>$

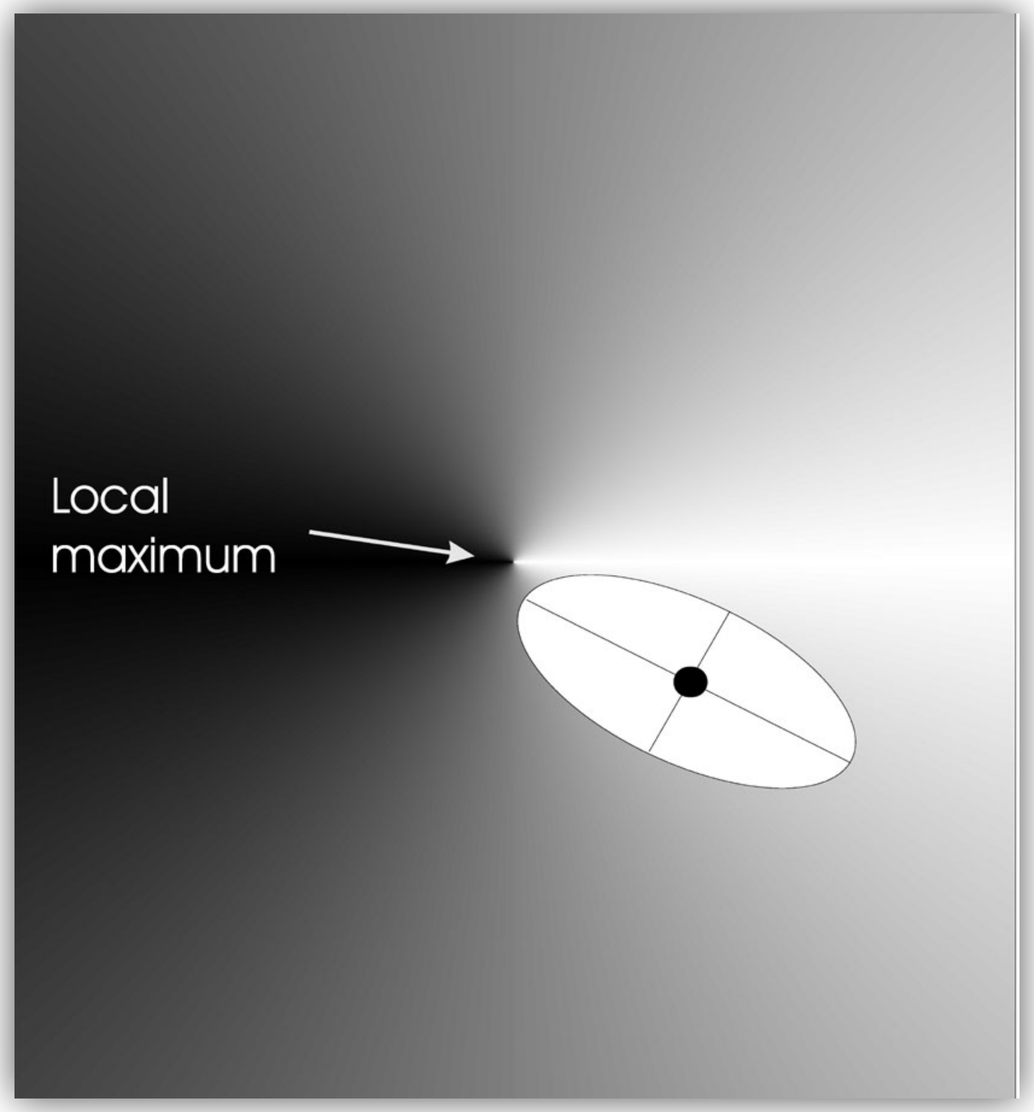
- so the mutation step size for each gene x_i will be different
- σ is mutated by multiplying it by an amount whose calculation contains *three* key terms:
  - a random value from N(0,1)
  - an overall learning rate τ, which applies to the whole genotype (as before)
  - a 'coordinate-wise' learning rate τ_i, which applies individually to each gene x_i
  - as before, the learning rates are set by the user

Real-valued Representation: Uncorrelated mutation with n σ values



- the perimeter of the elipse represents mutants with the same chance of being created
- allowing different mutation step sizes for each gene xi allows mutations to be more likely to happen along one axis
- which in this case would be more likely to steer the individual towards the local maximum

Real-valued Representation: Correlated Mutations



- correlated mutations allow the ellipses to have any orientation
- this is done by co-evolving a rotation parameter
- as we can see, this individual is more likely to mutate to a value close to the local maximum than the individual seen in the previous slide

Sophistication Comes at a Price

- as we increase the number of genes used to represent an individual we increase the size of the search space
- so the benefit of using a complex mutation operator could be outweighed by the time cost, or by the navigation difficulty of the more complex fitness landscape that it creates
- the benefit might also depend on the type of problem
- a common approach is to start with uncorrelated mutation with n σ values, and then:
  - if good results are obtained but at a slow pace then move to a simpler model (one σ, or fixed σ)
  - if the results aren't good enough then move to a more complex model (correlated mutation)

Real-valued Representation: Crossover Operators

discrete:

- each allele value in offspring z comes from one of its parents (x,y) with equal probability: $z_i = x_i \text{ or } y_i$
    - could use 1-point, n-point or uniform crossover

intermediate

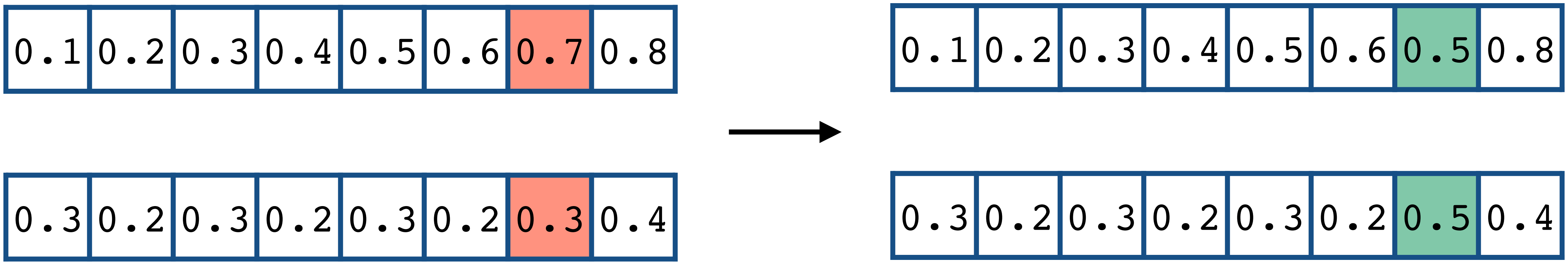
- exploits idea of creating children 'between' parents
    - *arithmetic recombination*
- $z_i = (1-\alpha).x_i + \alpha.y_i \text{ where } \alpha : 0 \leqslant \alpha \leqslant 1$
- parameter α can be:
    - constant: uniform arithmetical crossover
    - variable: for example, depend on the age of the population)
    - picked at random every time

Real-valued Representation: Single Arithmetic Crossover

- parents: $<x_1, \ldots, x_n>$ and $<y_1, \ldots, y_n>$
- pick a single gene (k) at random,
- first child is:

$$<x_1,\ldots,x_{k-1},\alpha.y_k,+(1-\alpha).x_k,\ldots,x_n>$$

- reverse x and y for second child
- example: $\alpha = 0.5$



0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
0.3	0.2	0.3	0.2	0.3	0.2	0.3	0.4

→

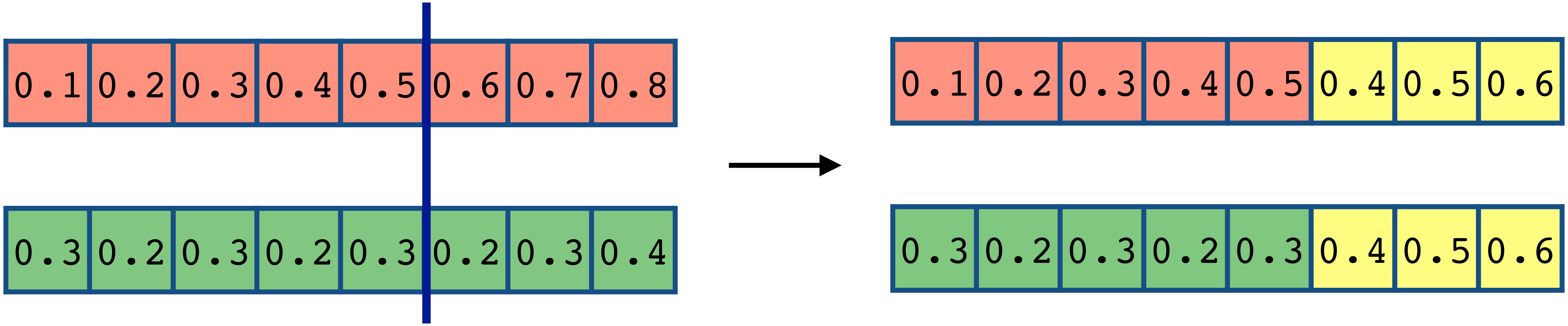
0.1	0.2	0.3	0.4	0.5	0.6	0.5	0.8
0.3	0.2	0.3	0.2	0.3	0.2	0.5	0.4

Real-valued Representation: Simple Arithmetic Crossover

- parents: $\langle x_1, \ldots, x_n \rangle$ and $\langle y_1, \ldots, y_n \rangle$
- pick a random gene (k) and after this point mix values
- first child is:

$$\langle x_1,\ldots,x_k,\alpha.y_{k+1},+(1-\alpha).x_{k+1},\ldots,\alpha.y_n,+(1-\alpha).x_n\rangle$$

- reverse x and y for second child
- example: $\alpha = 0.5$



0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
0.3	0.2	0.3	0.2	0.3	0.2	0.3	0.4

→

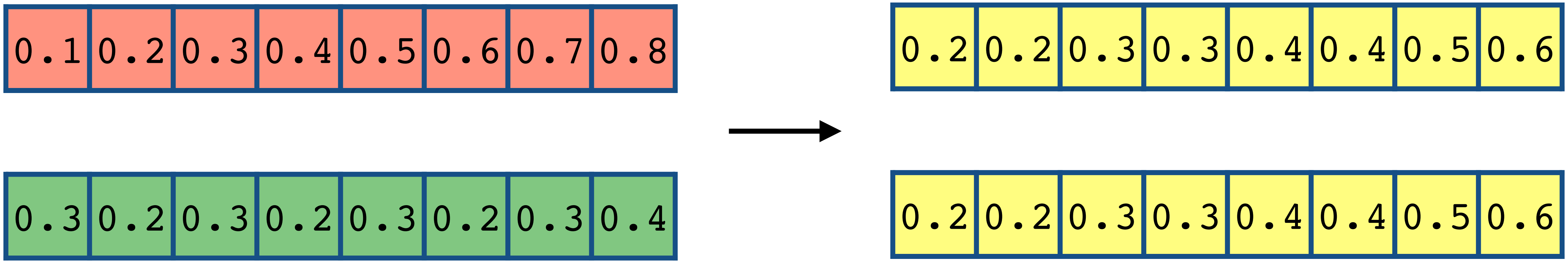
0.1	0.2	0.3	0.4	0.5	0.4	0.5	0.6
0.3	0.2	0.3	0.2	0.3	0.4	0.5	0.6

Real-valued Representation: Whole Arithmetic Crossover

- most common technique
- parents: $\langle x_1, \ldots, x_n \rangle$ and $\langle y_1, \ldots, y_n \rangle$
- first child is:

$$\langle \alpha . y_k, + (1-\alpha) . x_1, \ldots, y_n + (1-\alpha) . x_n \rangle$$

- reverse x and y for second child
- example: $\alpha = 0.5$



0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8

0.3	0.2	0.3	0.2	0.3	0.2	0.3	0.4

0.2	0.2	0.3	0.3	0.4	0.4	0.5	0.6

0.2	0.2	0.3	0.3	0.4	0.4	0.5	0.6

(because α = 0.5 both children are identical)

Real-valued Representation: Blend Arithmetic Crossover

- allows the generation of new values outside of the range $[x_i, y_i]$
- parents: $<x_1, \ldots, x_n>$ and $<y_1, \ldots, y_n>$
- assume that $x_i < y_i$
- then $d_i = y_i - x_i$
- x_i takes a new random value from the range $[x_i - \alpha d_i, y_i + \alpha d_i]$
- evidence suggests that the best results are achieved when $\alpha = 0.5$
- in words: when there’s the same probability of selecting a value from outside the range $[x_i, y_i]$ as there is of selecting a value from within that range

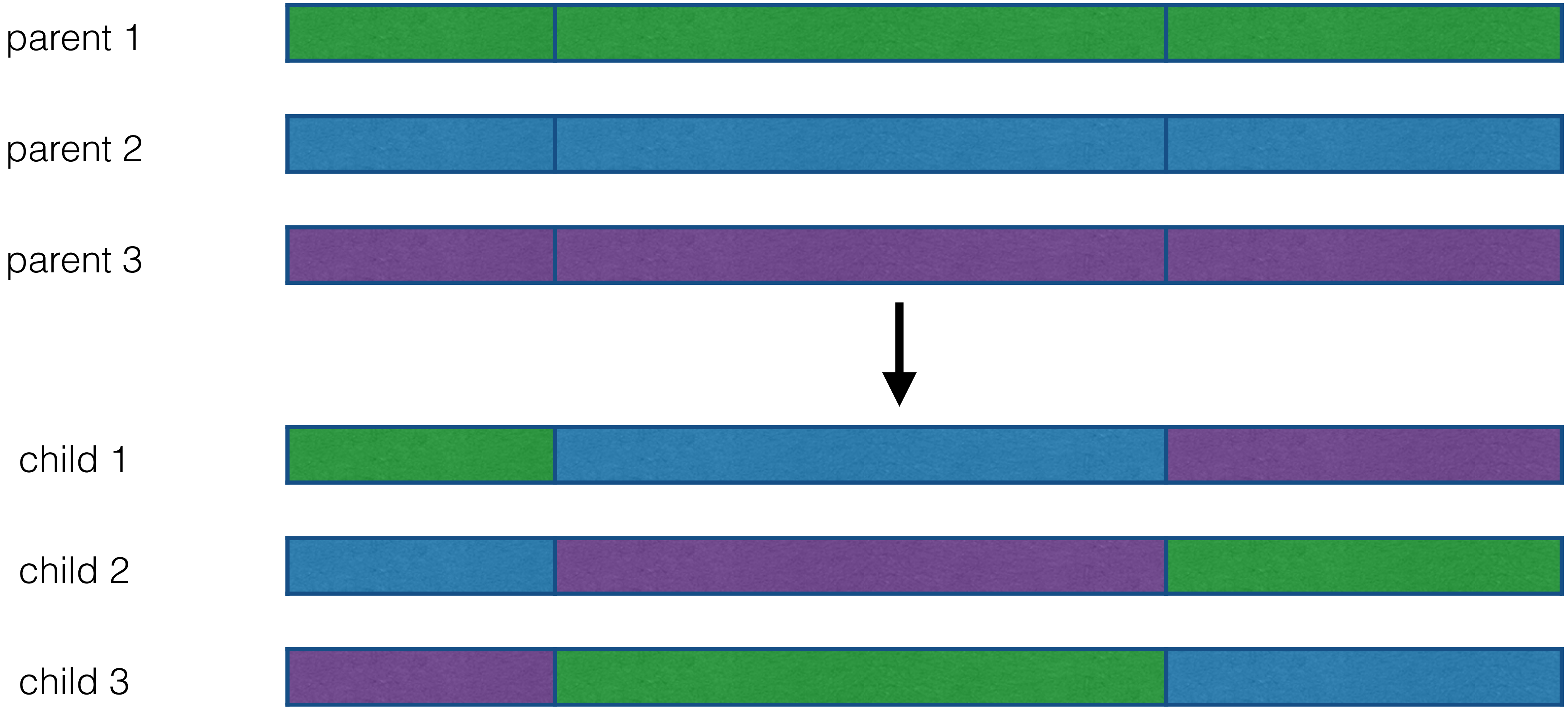
Real-valued Representation: Multi-Parent Recombination

- the representations and techniques used here are inspired by nature
- but they are not bound by it
- mutation uses n = 1 parent,
- ‘traditional’ crossover uses n = 2 parents
- so the extension to n > 2 parents is natural to examine
- the idea came about a long time ago (1960s)
- not often used, but has proven useful
- two main types have been used…

Real-valued Representation: Multi-Parent Recombination

segment and recombine:

- example: diagonal crossover for n parents:
- choose n-1 crossover points (same points in each parent)
- compose n children from the segments of each parent like this:



Real-valued Representation: Multi-Parent Recombination

arithmetical combination of alleles:

- example: arithmetic crossover for n parents:
  - i-th allele in the child is the average of the parents i-th allele values
- creates 'centre of mass' as child
- rarely used in genetic algorithms
- long known and used in evolution strategies
- many variations possible

Reading & References

- slides based on and adapted from,Chapter 4 (and slides) of Eiben & Smith's *Introduction to Evolutionary Computing*

- W.M. Spears: Evolutionary Algorithms: The Role of Mutation and Recombination, Springer 2000

- K. Deb: Representations. Part 4 of T. Bäck, D. Fogel and Z. Michalewicz (editors) Evolutionary Computation 1: Basic Algorithms and Operators, Institute of Physics Press

  - *note that above link leads directly to a .pdf download*