Evolutionary Computing

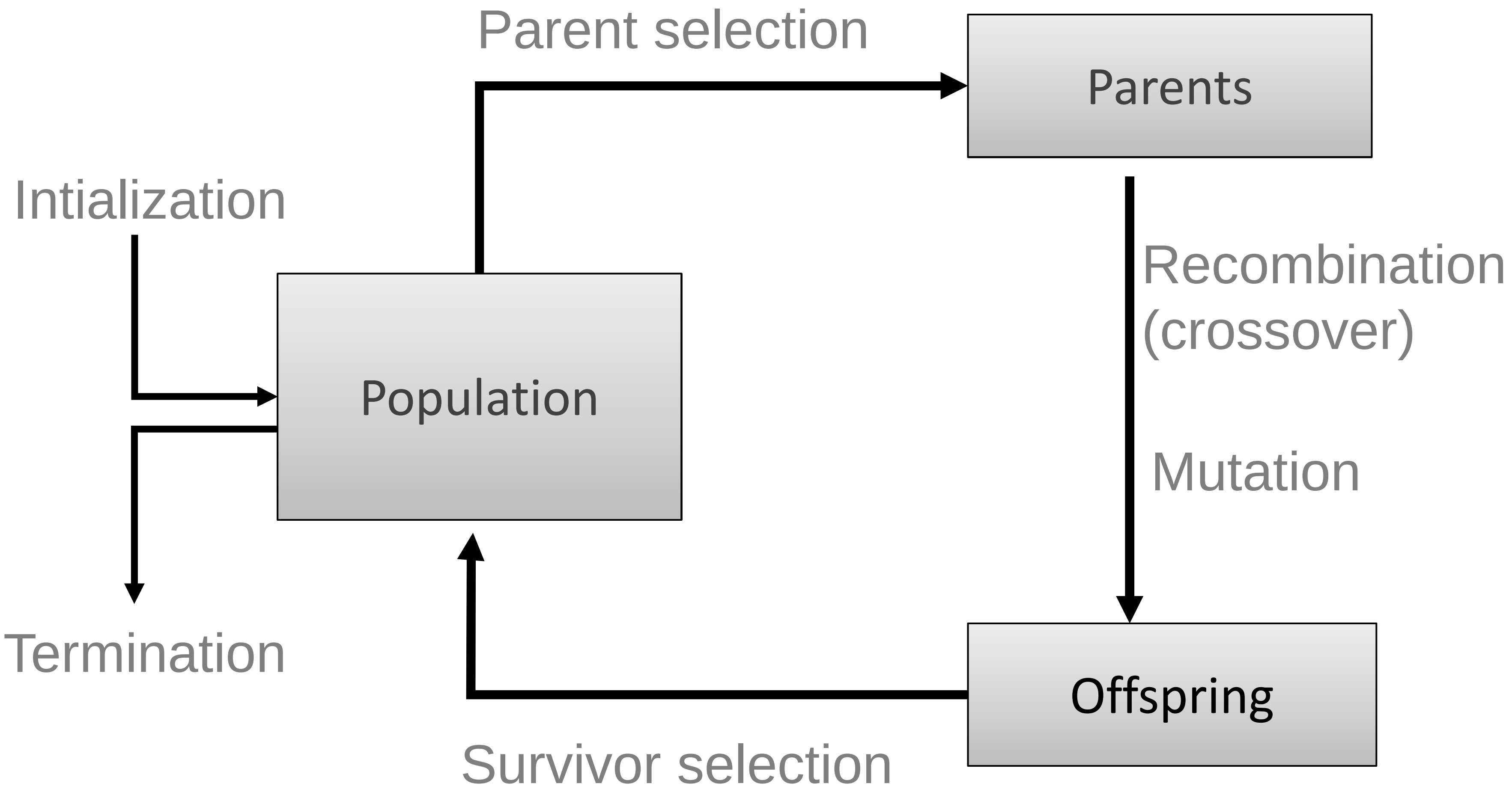
Chapter 4

Chapter 4:
Representation, Mutation, and Recombination

- Role of representation and variation operators.
- Most common representation of genomes:
  - Binary;
  - Integer;
  - Real-Valued or Floating-Point;
  - Permutation;
  - Tree.

Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Scheme of an EA:
General scheme of EAs



Parent selection

Parents

Intialization

Recombination (crossover)

Population

Mutation

Termination

Offspring

Survivor selection

Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

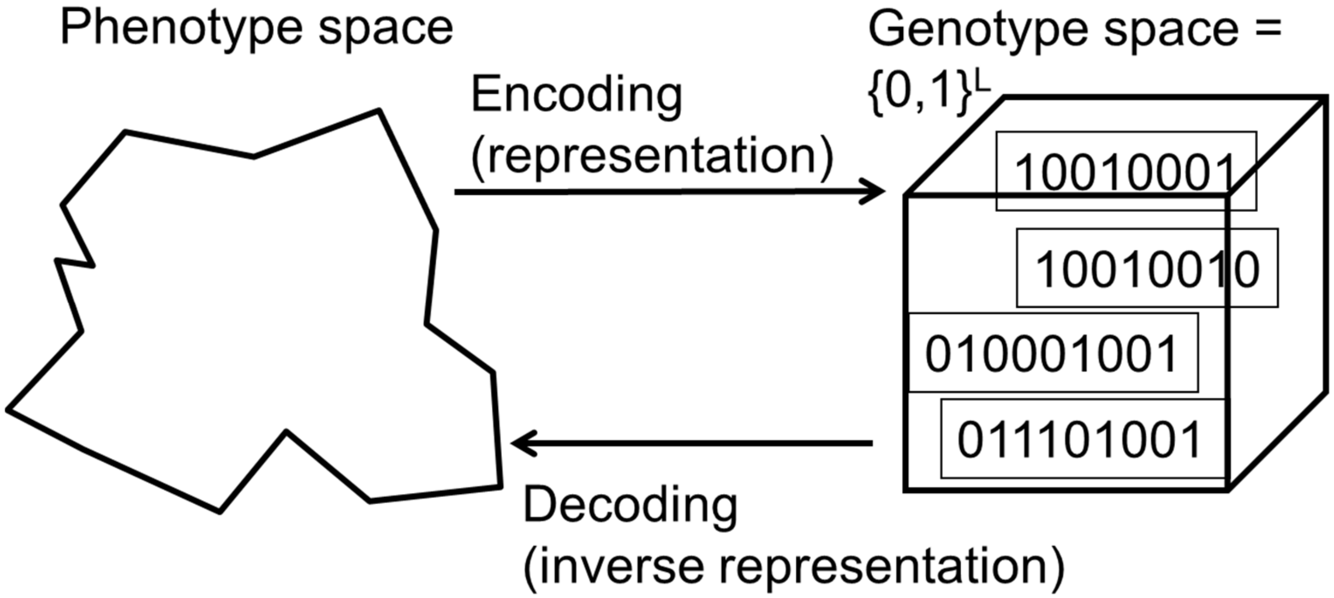
Role of representation and variation operators

- First stage of building an EA and most difficult one: choose *right* representation for the problem
- Variation operators: mutation and crossover
- Type of variation operators needed depends on chosen representation

- TSP problem
  - What are possible representations?

Binary Representation

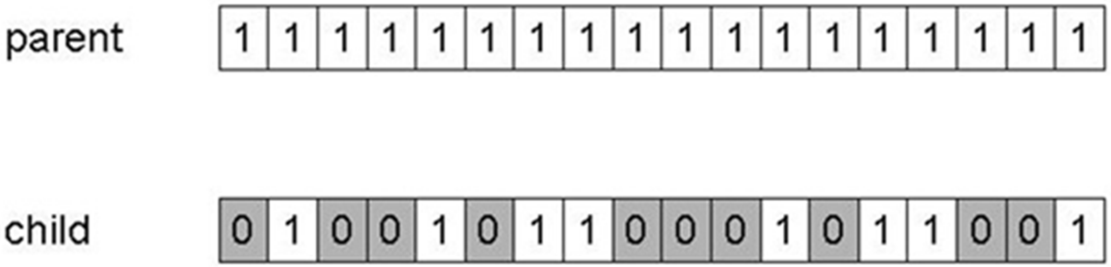
- One of the earliest representations
- Genotype consists of a string of binary digits



Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Binary Representation: Mutation

- Alter each gene independently with a probability p_m
- p_m is called the mutation rate
  - Typically between 1/pop_size and 1/ chromosome_length



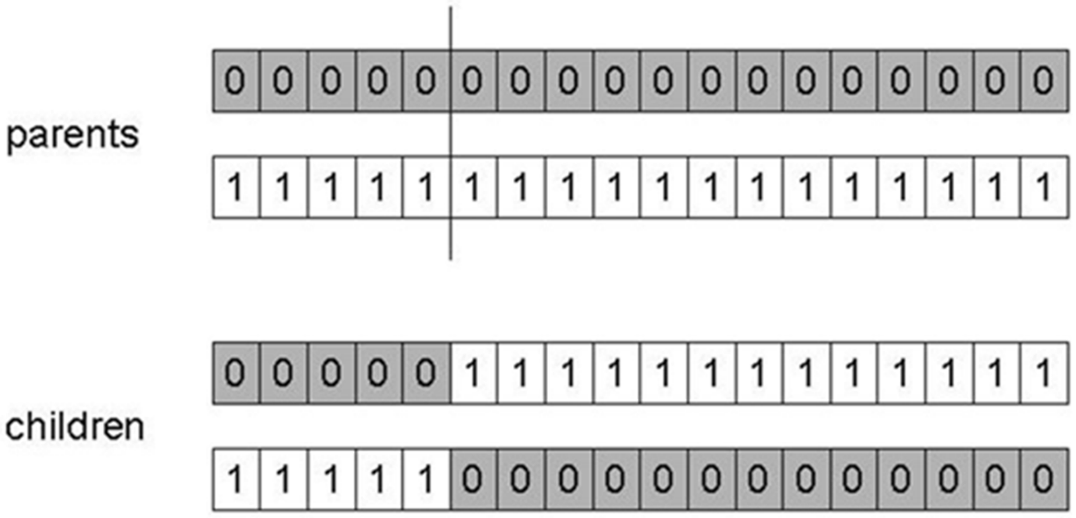
parent	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
child	0	1	0	0	1	0	1	1	0	0	0	1	0	1	1	0	0	1

- Mutation can cause variable effect (use gray coding)

Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Binary Representation:
1-point crossover

- Choose a random point on the two parents
- Split parents at this crossover point
- Create children by exchanging tails
- P_c typically in range (0.6, 0.9)



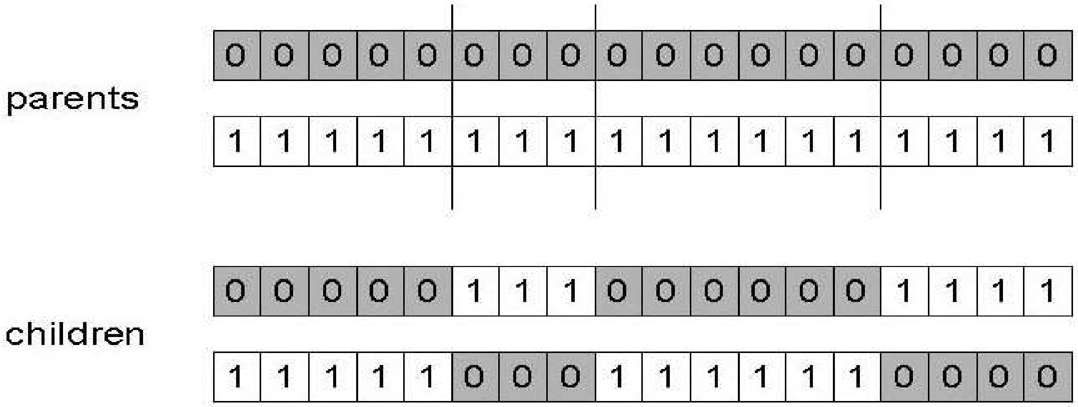
Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Binary Representation: Alternative Crossover Operators

- Why do we need other crossover(s)?
- Performance with 1-point crossover depends on the order that variables occur in the representation
  - More likely to keep together genes that are near each other
  - Can never keep together genes from opposite ends of string
  - This is known as Positional Bias
  - Can be exploited if we know about the structure of our problem, but this is not usually the case

Binary Representation:
n-point crossover

- Choose n random crossover points
- Split along those points
- Glue parts, alternating between parents
- Generalisation of 1-point (still some positional bias)



parents

0 0 0 0 0 | 0 0 0 | 0 0 0 0 0 0 | 0 0 0 0

1 1 1 1 1 | 1 1 1 | 1 1 1 1 1 1 | 1 1 1 1

children

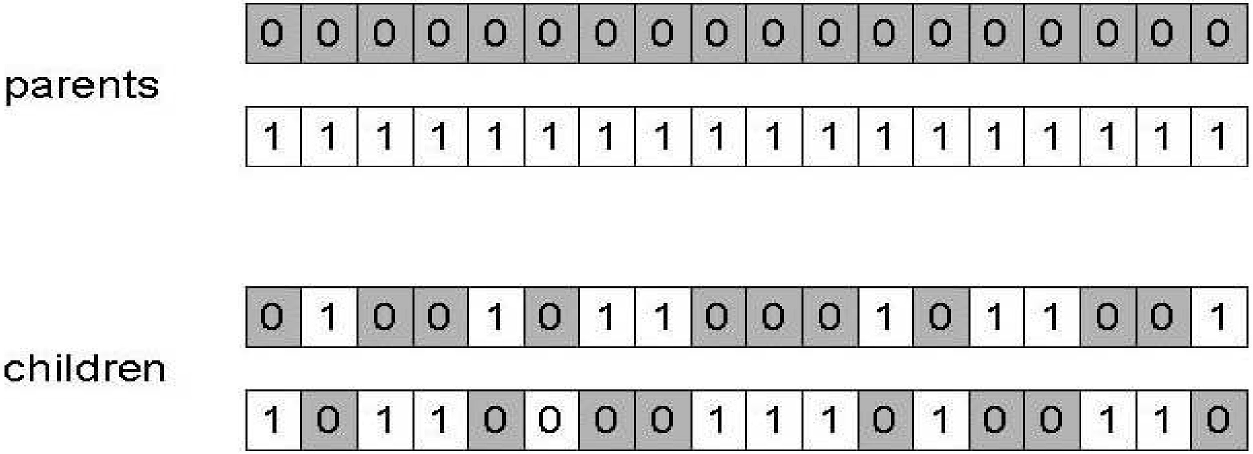
0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 1

1 1 1 1 1 0 0 0 1 1 1 1 1 1 0 0 0 0

Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Binary Representation: Uniform crossover

- Assign 'heads' to one parent, 'tails' to the other
- Flip a coin for each gene of the first child
- Make an inverse copy of the gene for the second child
- Inheritance is independent of position



parents

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

children

0 1 0 0 1 0 1 1 0 0 0 1 0 1 1 0 0 1

1 0 1 1 0 0 0 0 1 1 1 0 1 0 0 1 1 0

Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Binary Representation: Crossover OR mutation? (1/3)

- Decade long debate: which one is better / necessary / main-background
- Answer (at least, rather wide agreement):
  - it depends on the problem, but
  - in general, it is good to have both
  - both have another role
  - mutation-only-EA is possible, xover-only-EA would not work

Binary Representation: Crossover OR mutation? (2/3)

Exploration: Discovering promising areas in the search space, i.e. gaining information on the problem

Exploitation: Optimising within a promising area, i.e. using information

There is co-operation AND competition between them

- Crossover is explorative, it makes a *big* jump to an area somewhere "in between" two (parent) areas
- Mutation is exploitative, it creates random *small* diversions, thereby staying near (in the area of) the parent

Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Binary Representation: Crossover OR mutation? (3/3)

- Only crossover can combine information from two parents
- Only mutation can introduce new information (alleles)
- Crossover does not change the allele frequencies of the population (thought experiment: 50% 0’s on first bit in the population, ?% after performing n crossovers)
- To hit the optimum you often need a ‘lucky’ mutation

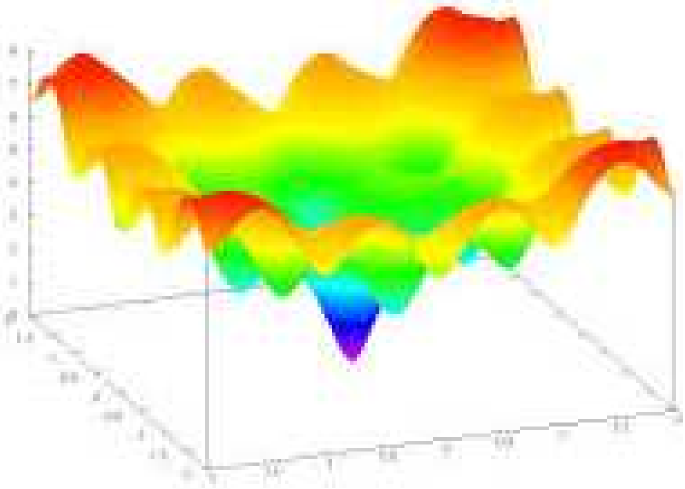
Integer Representation

- Nowadays it is generally accepted that it is better to encode numerical variables directly (integers, floating point variables)
- Some problems naturally have integer variables, e.g. image processing parameters
- Others take categorical values from a fixed set e.g. {blue, green, yellow, pink}
- N-point / uniform crossover operators work
- Extend bit-flipping mutation to make
  - “creep” i.e. more likely to move to similar value
    - Adding a small (positive or negative) value to each gene with probability $p_{.}$
  - Random resetting (esp. categorical variables)
    - With probability p_m a new value is chosen at random
- Same recombination as for binary representation

Real-Valued or Floating-Point Representation

- Many problems occur as real valued problems, e.g. continuous parameter optimisation $f : \Re^n \rightarrow \Re$
- Illustration: Ackley's function (often used in EC)

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



Real-Valued or Floating-Point Representation: Mapping real values on bit strings

$z \in [x,y] \subseteq \Re$ represented by $\{a_1,\ldots,a_L\} \in \{0,1\}^L$

- $[x,y] \rightarrow \{0,1\}^L$ must be invertible (one phenotype per genotype)
- $\Gamma: \{0,1\}^L \rightarrow [x,y]$ defines the representation

$$\Gamma(a_1,\ldots,a_L) = x + \frac{y-x}{2^L-1} \cdot \left(\sum_{j=0}^{L-1} a_{L-j} \cdot 2^j\right) \in [x,y]$$

- Only 2^L values out of infinite are represented
- L determines possible maximum precision of solution
- High precision → long chromosomes (slow evolution)

Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Real-Valued or Floating-Point Representation: Uniform Mutation

- General scheme of floating point mutations

$$\bar{x} = \langle x_1, \ \dots, x_l \rangle \rightarrow \bar{x}' = \langle x_1', \dots, x_l' \rangle$$

$$x_i, x_i' \in [LB_i, UB_i]$$

- Uniform Mutation

x_i' drawn randomly (uniform) from $[LB_i, UB_i]$

- Analogous to bit-flipping (binary) or random resetting (integers)

Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Real-Valued or Floating-Point Representation: Nonuniform Mutation

- Non-uniform mutations:
  - Many methods proposed, such as time-varying range of change etc.
  - Most schemes are probabilistic but usually only make a small change to value
  - Most common method is to add random deviate to each variable separately, taken from N(0, σ) Gaussian distribution and then curtail to range

    $x'_i = x_i + N(0,\sigma)$
  - Standard deviation σ, *mutation step size*, controls amount of change (2/3 of drawings will lie in range ($-\sigma$ to $+\sigma$))

Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Real-Valued or Floating-Point Representation: Self-Adaptive Mutation (1/2)

- Step-sizes are included in the genome and undergo variation and selection themselves: $\langle x_1,\ldots,x_n, \sigma \rangle$
- Mutation step size is not set by user but coevolves with solution
- Different mutation strategies may be appropriate in different stages of the evolutionary search process.

Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Real-Valued or Floating-Point Representation: Self-Adaptive Mutation (2/2)

- Mutate σ first
- Net mutation effect: ⟨ x, σ ⟩ → ⟨ x', σ' ⟩
- Order is important:
  - first σ → σ' (see later how)
  - then x → x' = x + N(0,σ')
- Rationale: new ⟨ x' ,σ' ⟩ is evaluated twice
  - Primary: x' is good if f(x') is good
  - Secondary: σ' is good if the x' it created is good
- Reversing mutation order this would not work

Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

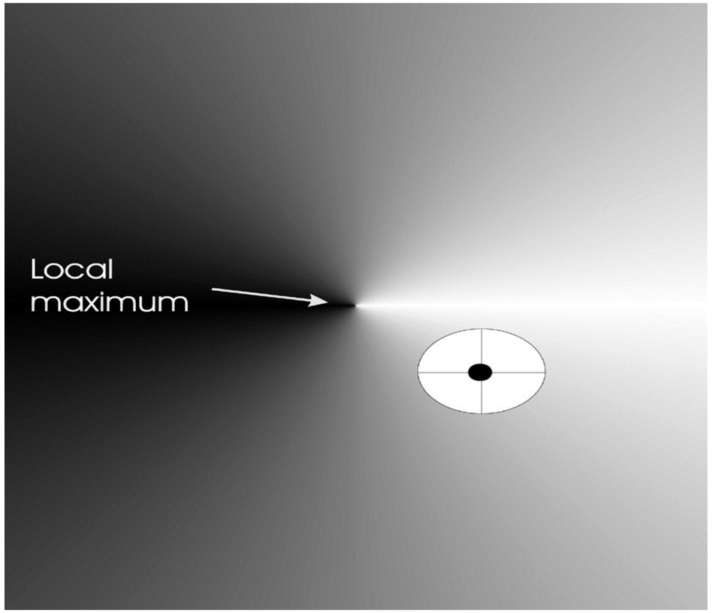
Real-Valued or Floating-Point Representation: Uncorrelated mutation with one σ (1/2)

- Chromosomes: $\langle x_1,\ldots,x_n, \sigma \rangle$
  - $\sigma' = \sigma \cdot \exp(\tau \cdot N(0,1))$
  - $x'_i = x_i + \sigma' \cdot N_i(0,1)$
- Typically the "learning rate" $\tau \propto 1/ n^{½}$
- And we have a boundary rule $\sigma' < \varepsilon_0 \Rightarrow \sigma' = \varepsilon_0$

Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Real-Valued or Floating-Point Representation: Uncorrelated mutation with one σ (2/2)

Mutants with equal likelihood



Circle: mutants having the same chance to be created

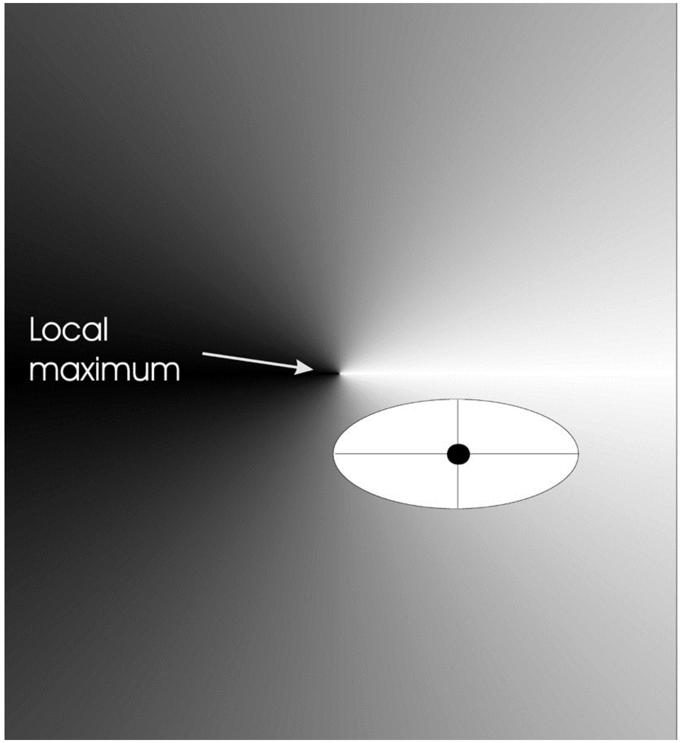
Real-Valued or Floating-Point Representation: Uncorrelated mutation with n σ's (1/2)

- Chromosomes: $\langle x_1,\dots,x_n, \sigma_1,\dots, \sigma_n \rangle$
  - $\sigma'_i = \sigma_i \cdot \exp(\tau' \cdot N(0,1) + \tau \cdot N_i(0,1))$
  - $x'_i = x_i + \sigma'_i \cdot N_i(0,1)$
- Two learning rate parameters:
  - τ' overall learning rate
  - τ coordinate wise learning rate
- $\tau' \propto 1/(2n)^{½}$ and $\tau \propto 1/(2n^{½})^{½}$
- Boundary rule: $\sigma_i' < \varepsilon_0 \Rightarrow \sigma_i' = \varepsilon_0$

Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Real-Valued or Floating-Point Representation: Uncorrelated mutation with n σ's (2/2)

Mutants with equal likelihood



Ellipse: mutants having the same chance to be created

Real-Valued or Floating-Point Representation: Correlated mutations (1/3)

- Chromosomes: $\langle x_1,\ldots,x_n, \sigma_1,\ldots, \sigma_n ,\alpha_1,\ldots, \alpha_k \rangle$
  where $k = n \bullet (n-1)/2$
- Covariance matrix C is defined as:
  - $c_{ii} = \sigma_i^2$
  - $c_{ij} = 0$ if i and j are not correlated
  - $c_{ij} = \frac{1}{2} \cdot (\sigma_i^2 - \sigma_j^2) \cdot \tan(2\,\alpha_{ij})$ if i and j are correlated
- Note the numbering / indices of the α's

Real-Valued or Floating-Point Representation: Correlated mutations (2/3)

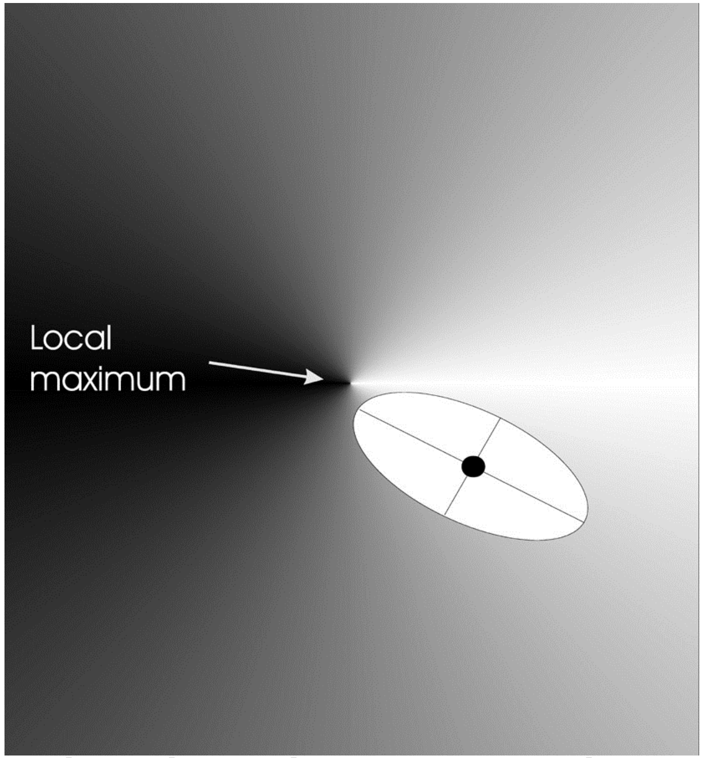
The mutation mechanism is then:

- $\sigma'_i = \sigma_i \cdot \exp(\tau' \cdot N(0,1) + \tau \cdot N_i(0,1))$
- $\alpha'_j = \alpha_j + \beta \cdot N(0,1)$
- $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{N}(\boldsymbol{0}, \boldsymbol{C}')$
  - **x** stands for the vector $\langle x_1, \ldots, x_n \rangle$
  - **C'** is the covariance matrix **C** after mutation of the α values
- $\tau \propto 1/(2n)^{½}$ and $\tau \propto 1/(2n^{½})^{½}$ and $\beta \approx 5°$
- $\sigma_i' < \varepsilon_0 \Rightarrow \sigma_i' = \varepsilon_0$ and
- $|\alpha'_j| > \pi \Rightarrow \alpha'_j = \alpha'_j - 2\pi\, \mathrm{sign}(\alpha'_j)$

- NB Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is probably the best EA for numerical optimisation, cf. CEC-2005 competition

Real-Valued or Floating-Point Representation: Correlated mutations (3/3)

Mutants with equal likelihood



Ellipse: mutants having the same chance to be created

Real-Valued or Floating-Point Representation: Crossover operators

- Discrete:
  - each allele value in offspring *z* comes from one of its parents *(x,y)* with equal probability: $z_i = x_i$ or y_i
  - Could use n-point or uniform
- Intermediate
  - exploits idea of creating children "between" parents (hence a.k.a. *arithmetic* recombination)
  - $z_i = \alpha x_i + (1 - \alpha) y_i$ where $\alpha : 0 \leq \alpha \leq 1.$
  - The parameter α can be:
    - constant: uniform arithmetical crossover
    - variable (e.g. depend on the age of the population)
    - picked at random every time

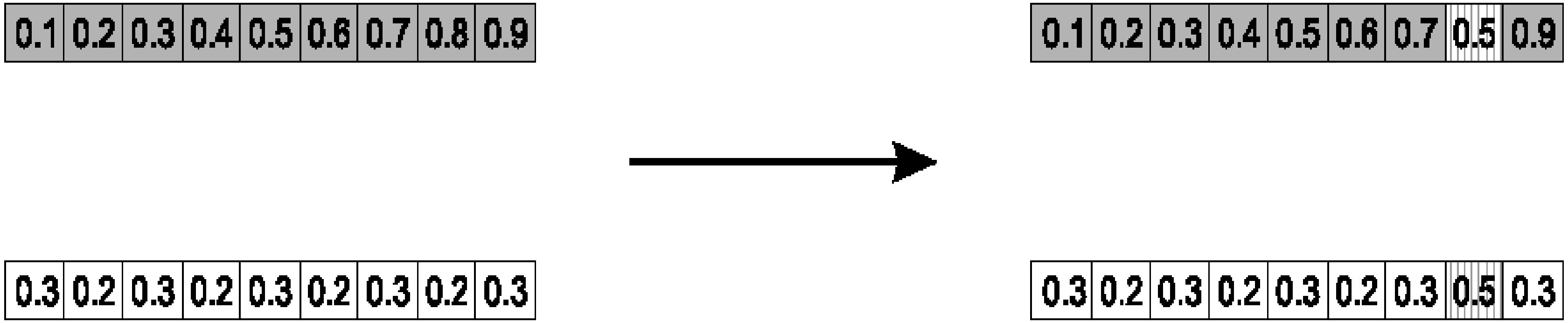
Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Real-Valued or Floating-Point Representation: Single arithmetic crossover

- Parents: ⟨$x_1,\ldots,x_n$⟩ and ⟨$y_1,\ldots,y_n$⟩
- Pick a single gene (k) at random,
- child$_1$ is:

$$\left\langle x_1, \ldots, x_k, \alpha \cdot y_k + (1-\alpha) \cdot x_k, \ldots, x_n \right\rangle$$

- Reverse for other child. e.g. with $\alpha = 0.5$



0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0.3	0.2	0.3	0.2	0.3	0.2	0.3	0.2	0.3

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.5	0.9
0.3	0.2	0.3	0.2	0.3	0.2	0.3	0.5	0.3

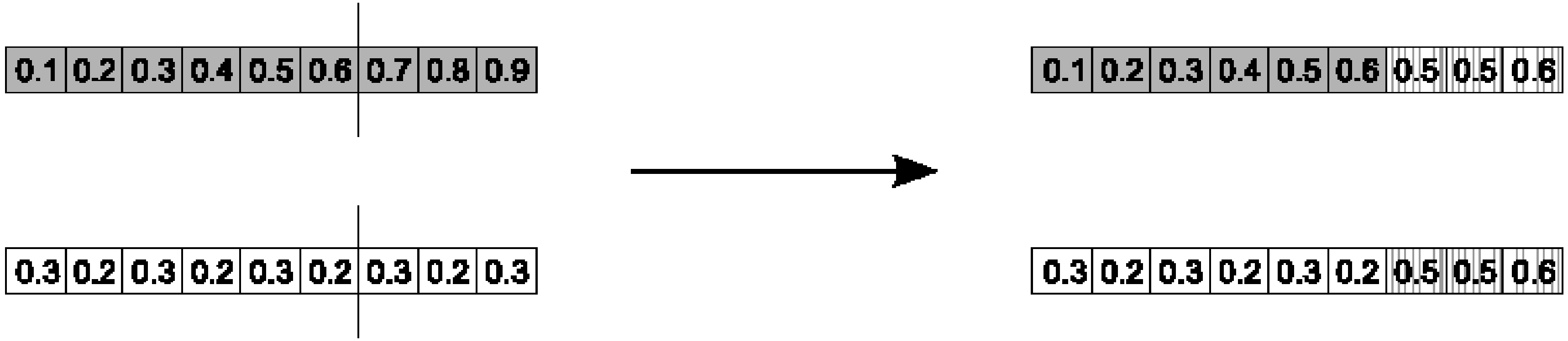
Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Real-Valued or Floating-Point Representation: Simple arithmetic crossover

- Parents: $\langle x_1,\dots,x_n \rangle$ and $\langle y_1,\dots,y_n \rangle$
- Pick a random gene (k) after this point mix values
- child$_1$ is:

$$\left\langle x_1, \dots, x_k, \alpha \cdot y_{k+1} + (1-\alpha)\cdot x_{k+1}, \dots, \alpha \cdot y_n + (1-\alpha)\cdot x_n \right\rangle$$

- reverse for other child. e.g. with $\alpha = 0.5$



0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0.3	0.2	0.3	0.2	0.3	0.2	0.3	0.2	0.3

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

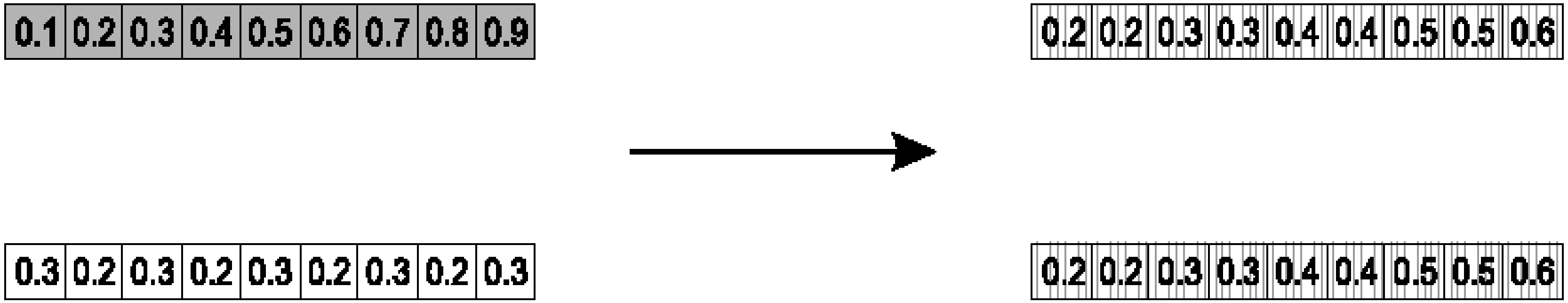
Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Real-Valued or Floating-Point Representation: Whole arithmetic crossover

- Most commonly used
- Parents: $\langle x_1,\ldots,x_n \rangle$ and $\langle y_1,\ldots,y_n\rangle$
- Child$_1$ is:

$$a \cdot \bar{x} + (1-a)\cdot \bar{y}$$

- reverse for other child. e.g. with $\alpha = 0.5$



0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9

0.3	0.2	0.3	0.2	0.3	0.2	0.3	0.2	0.3

0.2	0.2	0.3	0.3	0.4	0.4	0.5	0.5	0.6

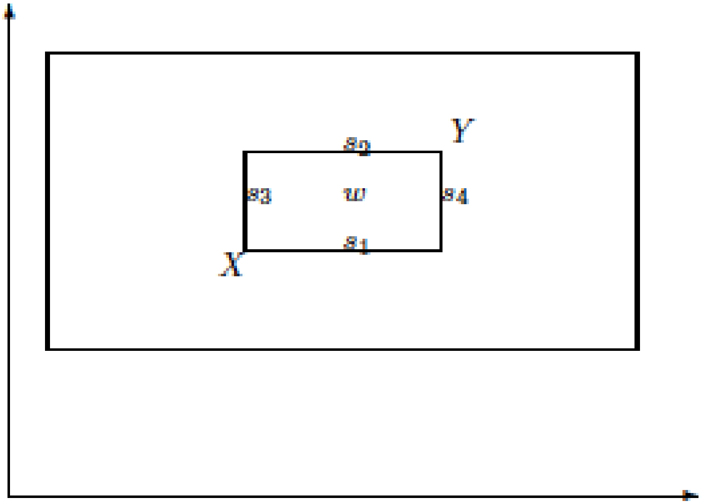
0.2	0.2	0.3	0.3	0.4	0.4	0.5	0.5	0.6

Real-Valued or Floating-Point Representation: Blend Crossover

- Parents: $\langle x_1,\ldots,x_n \rangle$ and $\langle y_1,\ldots,y_n \rangle$
- Assume $x_i < y_i$
- $d_i = y_i - x_i$
- Random sample $z_i = [x_i - \alpha d_i, x_i + \alpha d_i]$
- Original authors had best results with $\alpha = 0.5$

Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Real-Valued or Floating-Point Representation: Overview different possible offspring



- Single arithmetic:
- $\{s_1, s_2, s_3, s_4\}$
- Simple arithmetic / whole arithmetic:
- inner box (w = alpha 0.5)
- Blend crossover:
- outer box

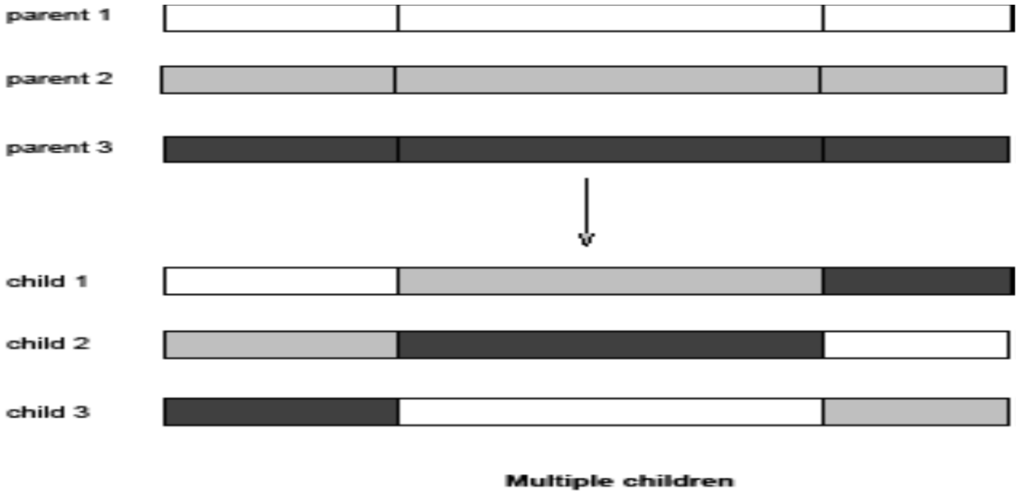
Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Real-Valued or Floating-Point Representation: Multi-parent recombination

- Recall that we are not constricted by the practicalities of nature
- Noting that mutation uses $n = 1$ parent, and “traditional” crossover $n = 2$, the extension to $n > 2$ is natural to examine
- Been around since 1960s, still rare but studies indicate useful

Real-Valued or Floating-Point Representation: Multi-parent recombination, type 1

- Idea: segment and recombine parents
- Example: diagonal crossover for *n* parents:
  - Choose *n-1* crossover points (same in each parent)
  - Compose *n* children from the segments of the parents in along a “diagonal”, wrapping around



- This operator generalises 1-point crossover

Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Real-Valued or Floating-Point Representation: Multi-parent recombination, type 2

- Idea: arithmetical combination of (real valued) alleles
- Example: arithmetic crossover for *n* parents:
  - *i*-th allele in child is the average of the parents' *i*-th alleles
- Creates center of mass as child
- Odd in genetic algorithms, long known and used in evolution strategies

Permutation Representations

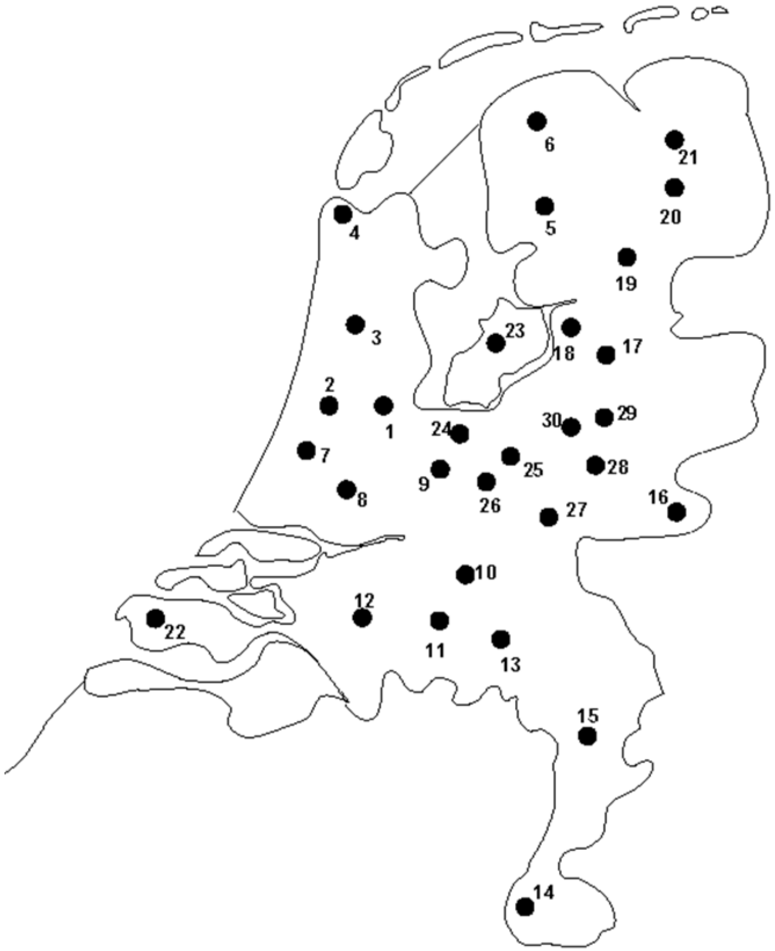
- Ordering/sequencing problems form a special type
- Task is (or can be solved by) arranging some objects in a certain order
  - Example: production scheduling: important thing is which elements are scheduled before others (order)
  - Example: Travelling Salesman Problem (TSP) : important thing is which elements occur next to each other (adjacency)
- These problems are generally expressed as a permutation:
  - if there are n variables then the representation is as a list of n integers, each of which occurs exactly once

Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Permutation Representation: TSP example

- Problem:
  - Given n cities
  - Find a complete tour with minimal length
- Encoding:
  - Label the cities 1, 2, … , *n*
  - One complete tour is one permutation (e.g. for n =4 [1,2,3,4], [3,4,2,1] are OK)
- Search space is BIG:

  for 30 cities there are 30! ≈ 10^{32} possible tours



Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Permutation Representations: Mutation

- Normal mutation operators lead to inadmissible solutions
    - e.g. bit-wise mutation: let gene *i* have value *j*
    - changing to some other value *k* would mean that *k* occurred twice and *j* no longer occurred
- Therefore must change at least two values
- Mutation parameter now reflects the probability that some operator is applied once to the whole string, rather than individually in each position

Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Permutation Representations:
Swap mutation

- Pick two alleles at random and swap their positions

Permutation Representations: Insert Mutation

- Pick two allele values at random
- Move the second to follow the first, shifting the rest along to accommodate
- Note that this preserves most of the order and the adjacency information

Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Permutation Representations: Scramble mutation

- Pick a subset of genes at random
- Randomly rearrange the alleles in those positions

Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

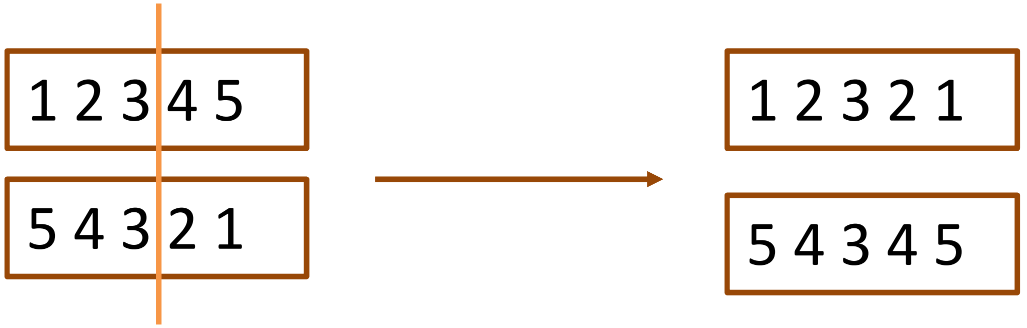
Permutation Representations: Inversion mutation

- Pick two alleles at random and then invert the substring between them.
- Preserves most adjacency information (only breaks two links) but disruptive of order information

Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Permutation Representations: Crossover operators

- “Normal” crossover operators will often lead to inadmissible solutions



1 2 3 4 5

5 4 3 2 1

1 2 3 2 1

5 4 3 4 5

- Many specialised operators have been devised which focus on combining order or adjacency information from the two parents

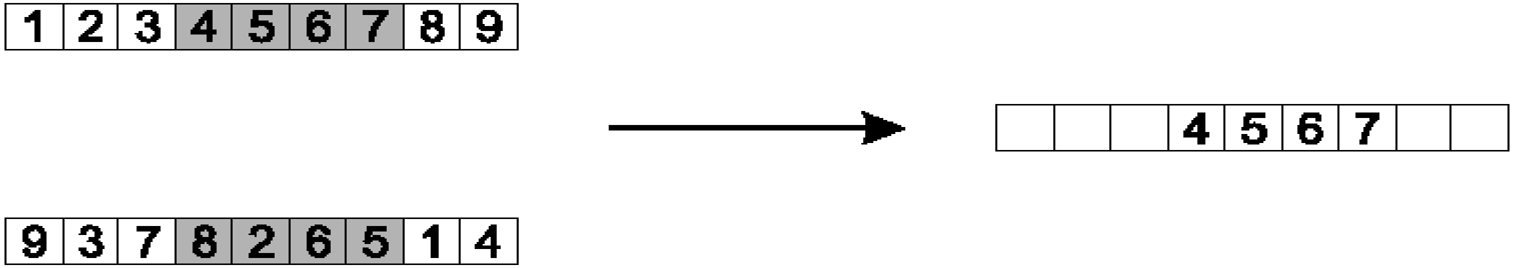
Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Permutation Representations: Order 1 crossover (1/2)

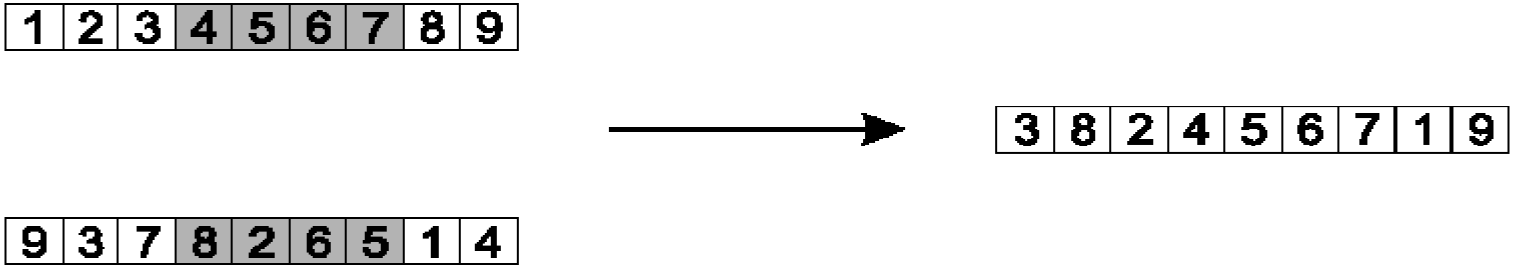
- Idea is to preserve relative order that elements occur
- Informal procedure:
  - 1. Choose an arbitrary part from the first parent
  - 2. Copy this part to the first child
  - 3. Copy the numbers that are not in the first part, to the first child:
    - starting right from cut point of the copied part,
    - using the **order** of the second parent
    - and wrapping around at the end
  - 4. Analogous for the second child, with parent roles reversed

Permutation Representations:
Order 1 crossover (2/2)

- Copy randomly selected set from first parent



- Copy rest from second parent in order 1,9,3,8,2



Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Permutation Representations: Partially Mapped Crossover (PMX) (1/2)

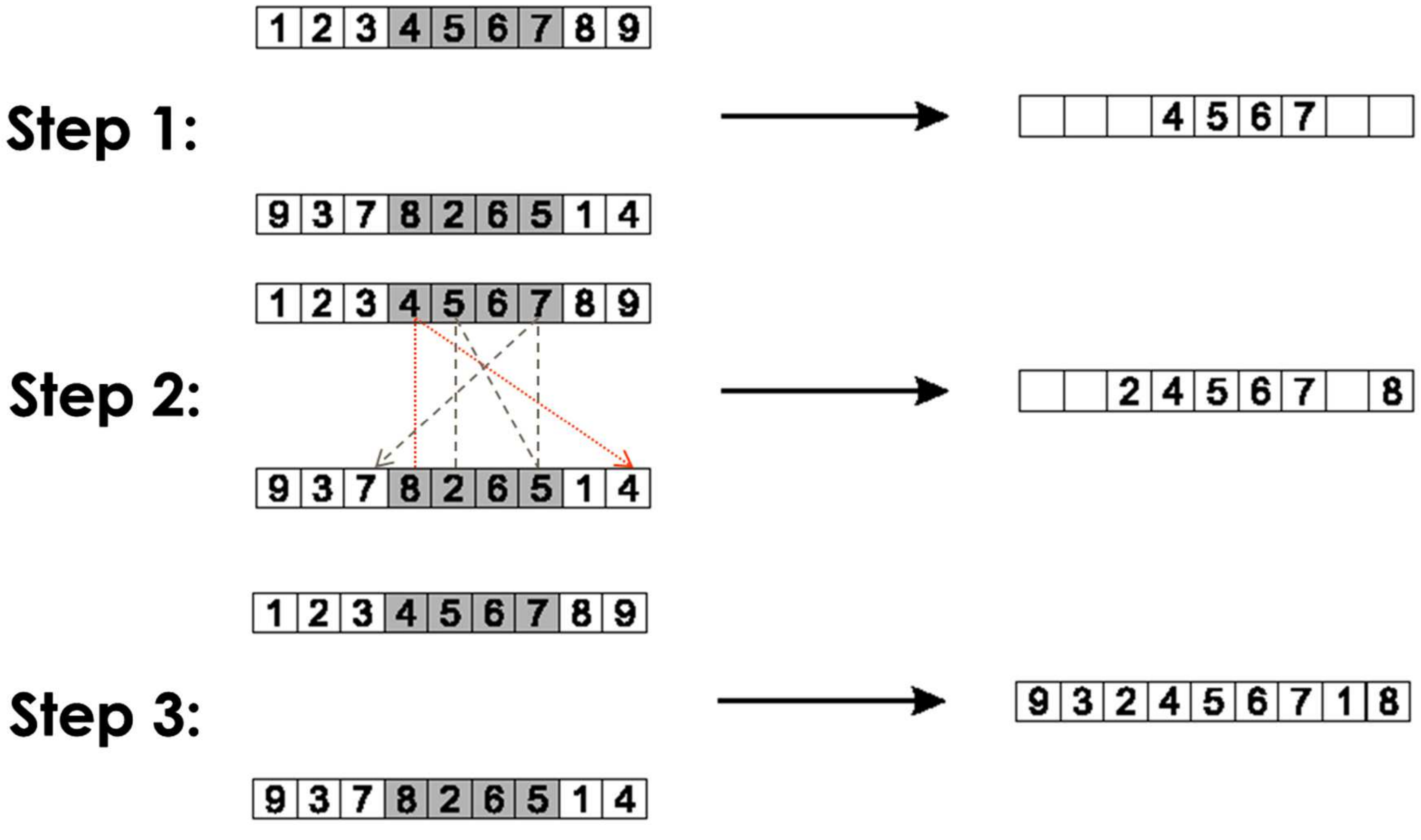
Informal procedure for parents P1 and P2:

1. Choose random segment and copy it from P1
2. Starting from the first crossover point look for elements in that segment of P2 that have not been copied
3. For each of these *i* look in the offspring to see what element *j* has been copied in its place from P1
4. Place *i* into the position occupied *j* in P2, since we know that we will not be putting *j* there (as is already in offspring)
5. If the place occupied by *j* in P2 has already been filled in the offspring *k*, put *i* in the position occupied by *k* in P2
6. Having dealt with the elements from the crossover segment, the rest of the offspring can be filled from P2.

Second child is created analogously

Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Permutation Representations: Partially Mapped Crossover (PMX) (2/2)



Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Permutation Representations: Cycle crossover (1/2)

Basic idea:

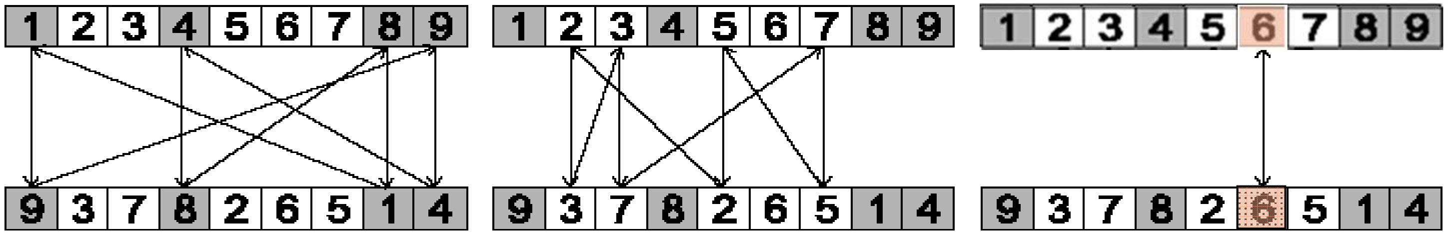
Each allele comes from one parent *together with its position*.

Informal procedure:

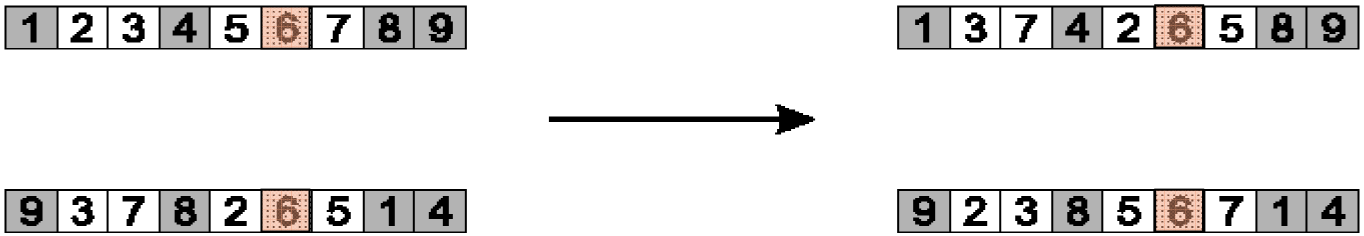
1. Make a cycle of alleles from P1 in the following way.
    - (a) Start with the first allele of P1.
    - (b) Look at the allele at the *same position* in P2.
    - (c) Go to the position with the *same allele* in P1.
    - (d) Add this allele to the cycle.
    - (e) Repeat step b through d until you arrive at the first allele of P1.
2. Put the alleles of the cycle in the first child on the positions they have in the first parent.
3. Take next cycle from second parent

Permutation Representations: Cycle crossover (2/2)

- Step 1: identify cycles



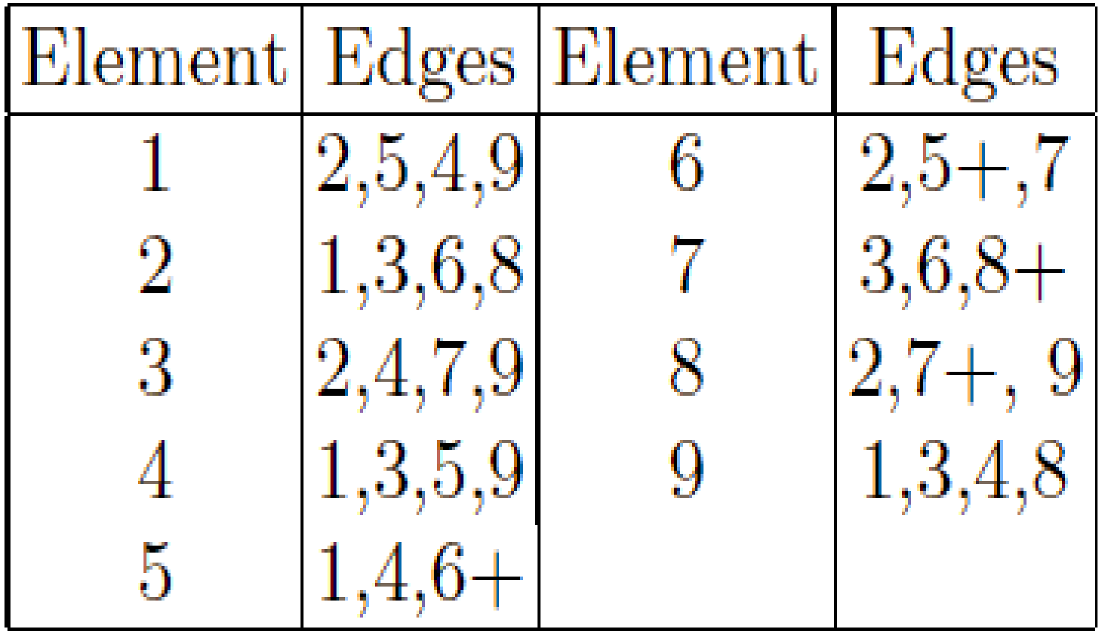
- Step 2: copy alternate cycles into offspring



Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Permutation Representations: Edge Recombination (1/3)

- Works by constructing a table listing which edges are present in the two parents, if an edge is common to both, mark with a +
- e.g. [1 2 3 4 5 6 7 8 9] and [9 3 7 8 2 6 5 1 4]



Element	Edges	Element	Edges
1	2,5,4,9	6	2,5+,7
2	1,3,6,8	7	3,6,8+
3	2,4,7,9	8	2,7+, 9
4	1,3,5,9	9	1,3,4,8
5	1,4,6+		

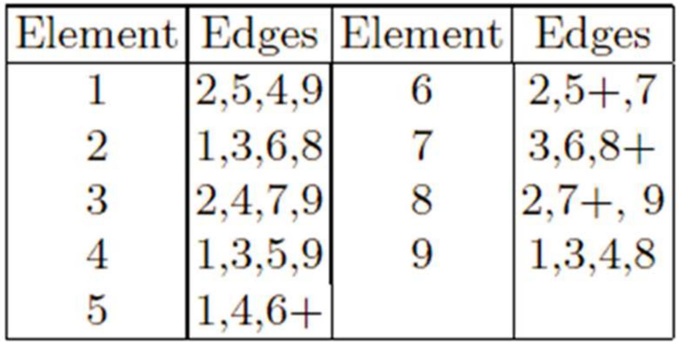
Permutation Representations: Edge Recombination (2/3)

Informal procedure: once edge table is constructed

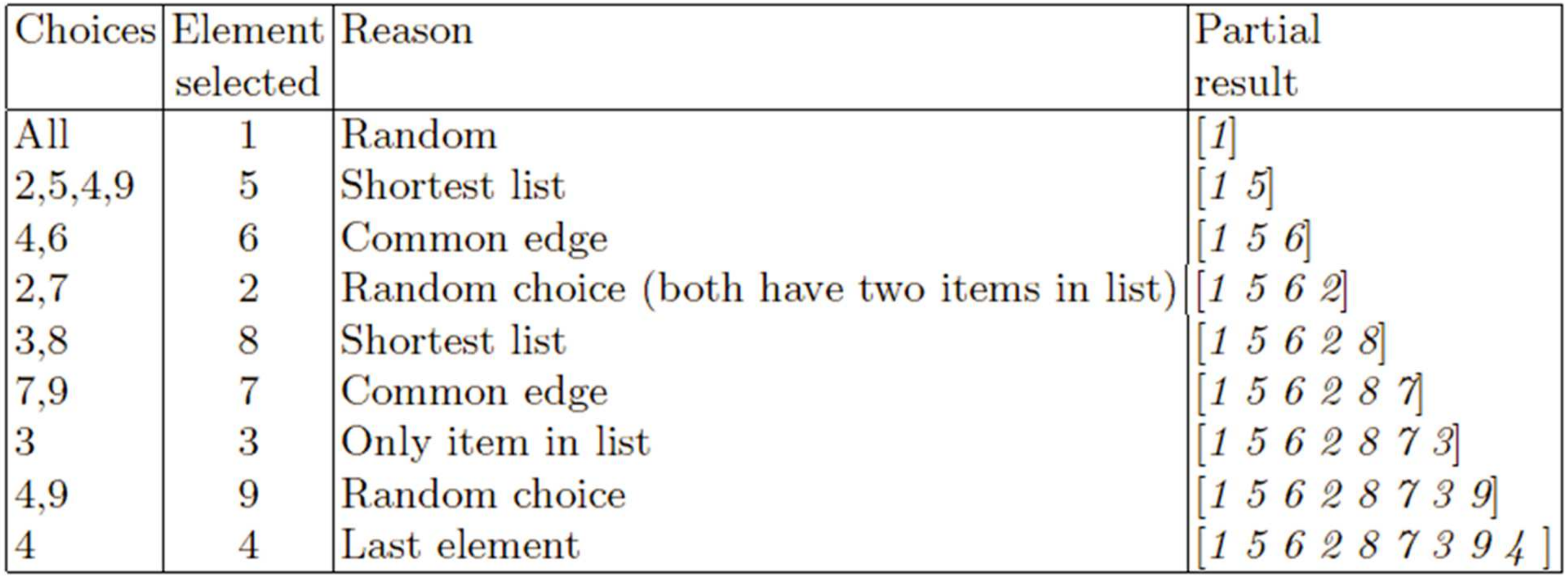
1. Pick an initial element, *entry*, at random and put it in the offspring
2. Set the variable *current element = entry*
3. Remove all references to *current element* from the table
4. Examine list for current element:
    - If there is a common edge, pick that to be next element
    - Otherwise pick the entry in the list which itself has the shortest list
    - Ties are split at random
5. In the case of reaching an empty list:
    - a new element is chosen at random

Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Permutation Representations: Edge Recombination (3/3)



Element	Edges	Element	Edges
1	2,5,4,9	6	2,5+,7
2	1,3,6,8	7	3,6,8+
3	2,4,7,9	8	2,7+, 9
4	1,3,5,9	9	1,3,4,8
5	1,4,6+		



Choices	Element selected	Reason	Partial result
All	1	Random	[1]
2,5,4,9	5	Shortest list	[1 5]
4,6	6	Common edge	[1 5 6]
2,7	2	Random choice (both have two items in list)	[1 5 6 2]
3,8	8	Shortest list	[1 5 6 2 8]
7,9	7	Common edge	[1 5 6 2 8 7]
3	3	Only item in list	[1 5 6 2 8 7 3]
4,9	9	Random choice	[1 5 6 2 8 7 3 9]
4	4	Last element	[1 5 6 2 8 7 3 9 4]

Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

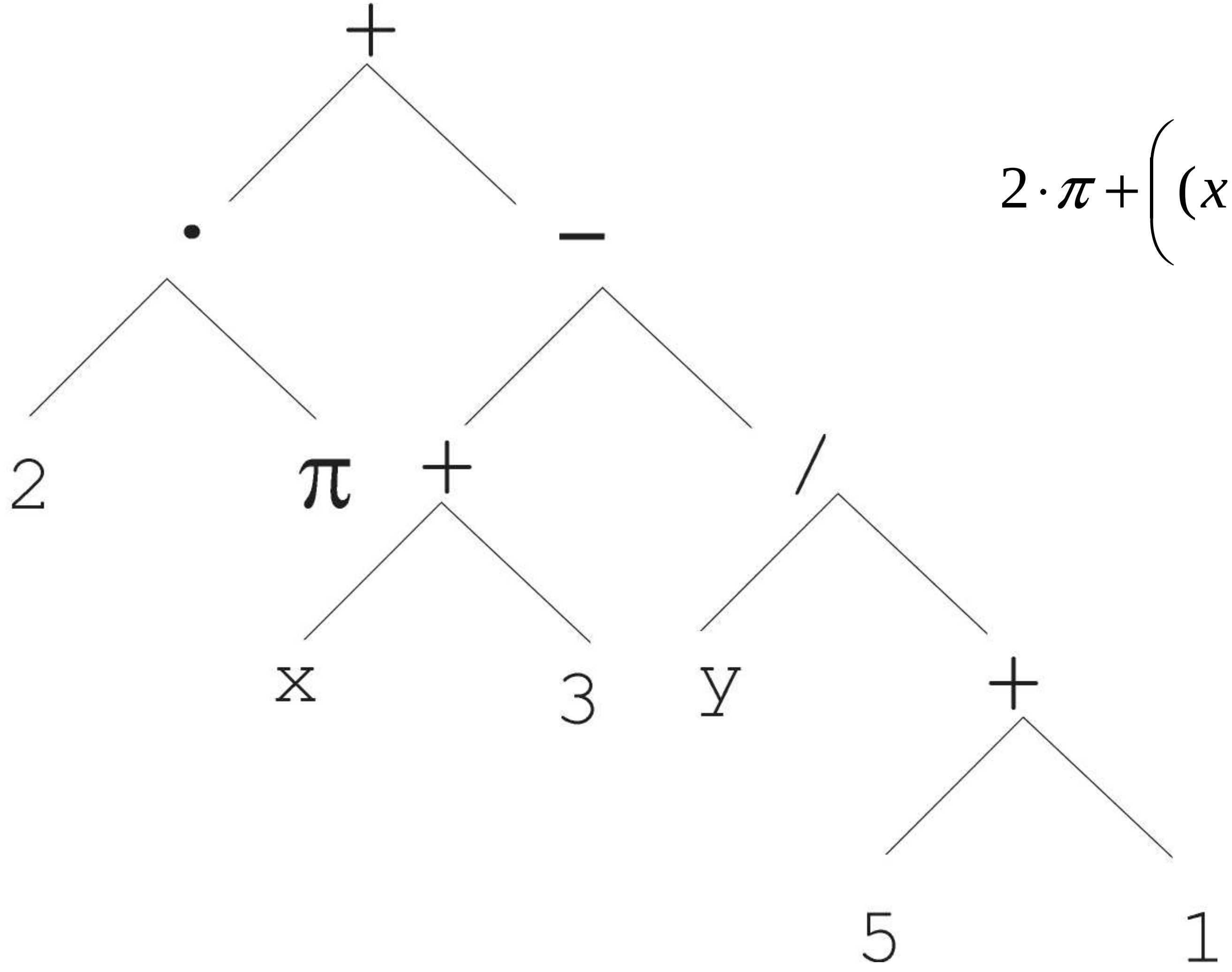
Tree Representation (1/6)

- Trees are a universal form, e.g. consider
- Arithmetic formula: $2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$
- Logical formula: $(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$
- Program:

```
i =1;
while (i < 20)
{
        i = i +1
}
```

Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

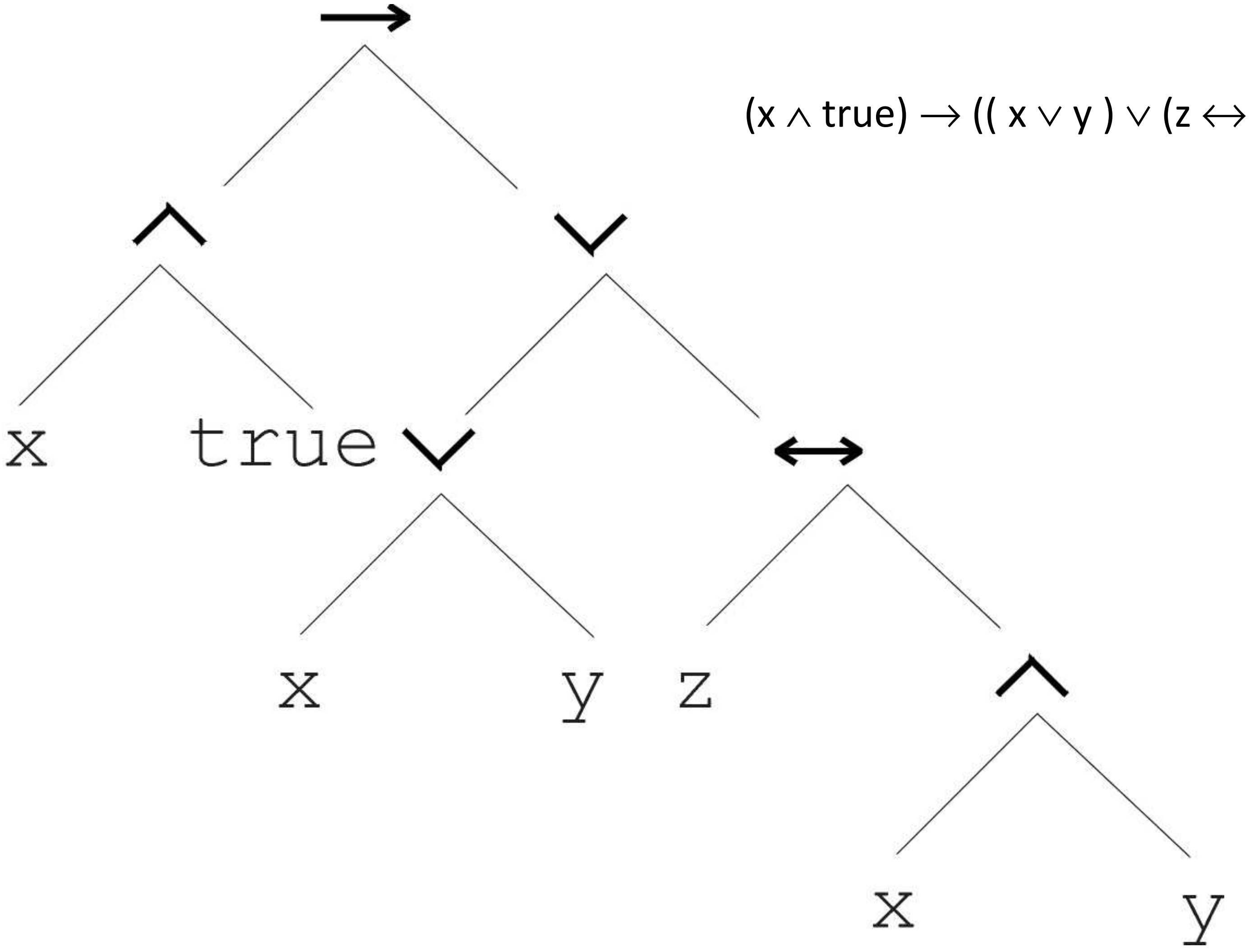
Tree Representation (2/6)



$$2\cdot\pi+\left((x+3)-\frac{y}{5+1}\right)$$

Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

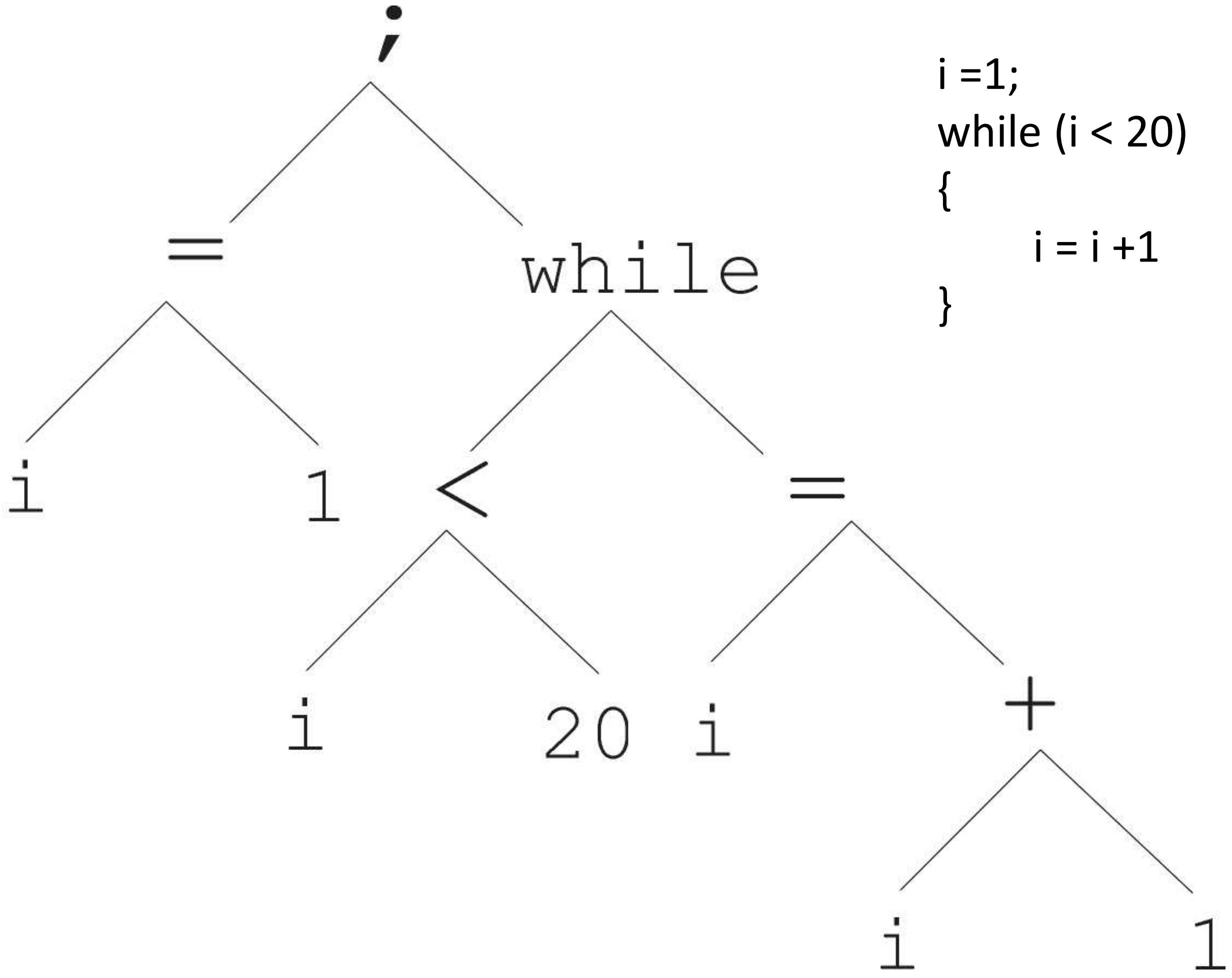
Tree Representation (3/6)



$(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$

Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Tree Representation (4/6)



```
i =1;
while (i < 20)
{
        i = i +1
}
```

Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Tree Representation (5/6)

- In GA, ES, EP chromosomes are linear structures (bit strings, integer string, real-valued vectors, permutations)
- Tree shaped chromosomes are non-linear structures
- In GA, ES, EP the size of the chromosomes is fixed
- Trees in GP may vary in depth and width

Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

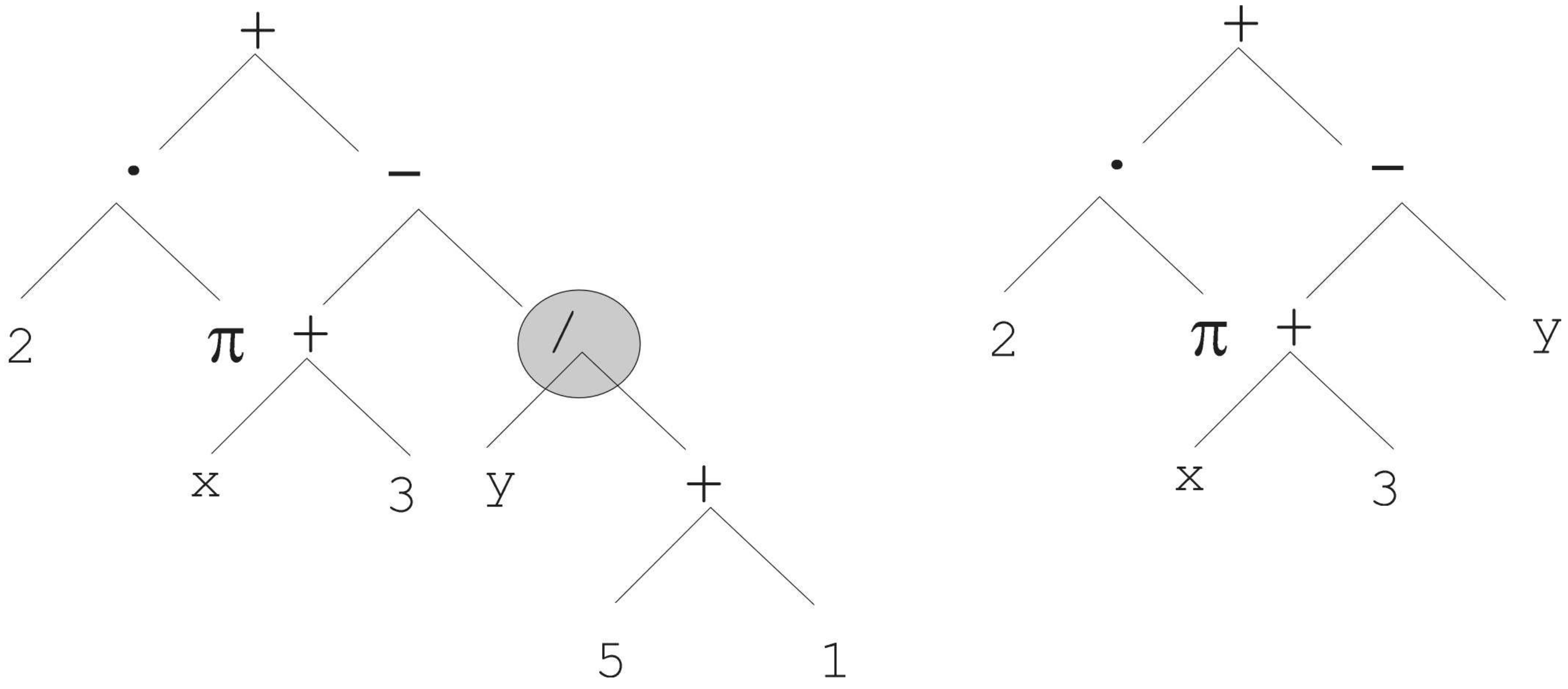
Tree Representation (6/6)

- Symbolic expressions can be defined by
  - Terminal set T
  - Function set F (with the arities of function symbols)
- Adopting the following general recursive definition:
  - Every $t \in T$ is a correct expression
  - $f(e_1, \ldots, e_n)$ is a correct expression if $f \in F$, arity(f)=n and $e_1, \ldots, e_n$ are correct expressions
  - There are no other forms of correct expressions
- In general, expressions in GP are not typed (closure property: any $f \in F$ can take any $g \in F$ as argument)

Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Tree Representation: Mutation (1/2)

- Most common mutation: replace randomly chosen subtree by randomly generated tree



Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

Tree Representation: Mutation (2/2)

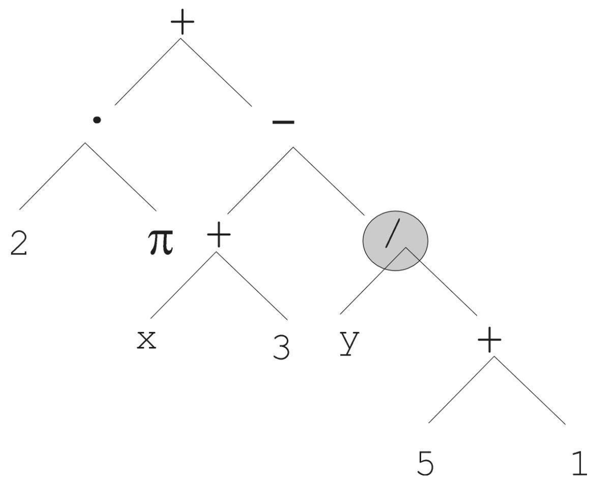
- Mutation has two parameters:
  - Probability p_m to choose mutation
  - Probability to chose an internal point as the root of the subtree to be replaced
- Remarkably p_m is advised to be 0 (Koza'92) or very small, like 0.05 (Banzhaf et al. '98)
- The size of the child can exceed the size of the parent

Tree Representation: Recombination (1/2)

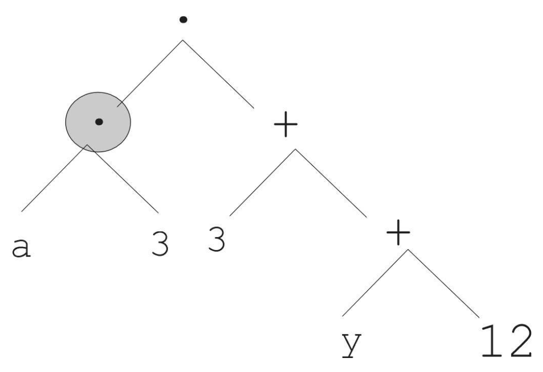
- Most common recombination: exchange two randomly chosen subtrees among the parents
- Recombination has two parameters:
  - Probability p_c to choose recombination
  - Probability to chose an internal point within each parent as crossover point
- The size of offspring can exceed that of the parents

Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014

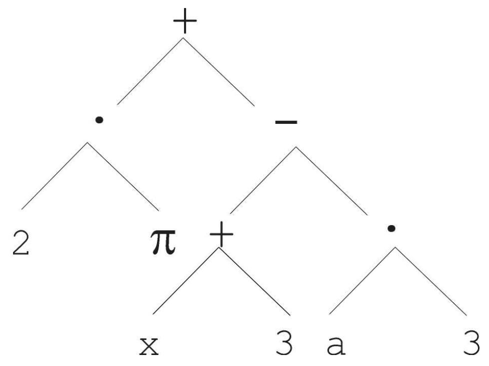
Tree Representation: Recombination (2/2)



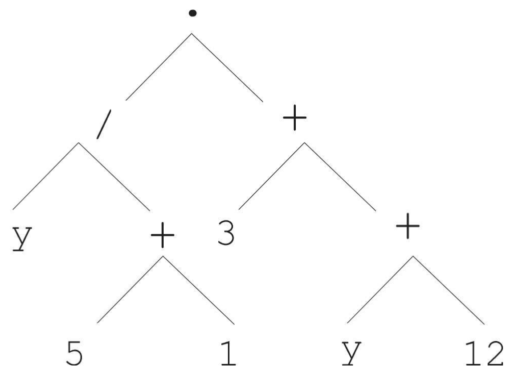
Parent 1



Parent 2



Child 1



Child 2

Adapted from A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing 2014