UiO : University of Oslo

institutt for informatikk

IN3050/IN4050 - Introduction to Artificial Intelligence and Machine Learning

Evolutionary Algorithms – Introduction and Representations

Pooya Zakeri

Motivation (1/2)

- Early attempts at **classical AI** (rule-based systems, symbolic reasoning) failed to replicate complex human intelligence.
- **Non-classical approaches**, while innovative, struggled to handle tasks that required flexibility and adaptability.
- **Two prominent fields emerged in response:**
  1. **Connectionism** *(Neural Networks, Parallel Processing)*
     - Focused on **neural networks** and **parallel processing**, modeling intelligence as a distributed system of nodes that can learn and adapt.
     - Inspired by how the human brain processes information through interconnected neurons.
     - Connectionism contributed to the rise of modern **machine learning** and **deep neural networks**.
  2. **Evolutionary Computing** *(Inspired by Biological Evolution) cont…*

Motivation (1/2)

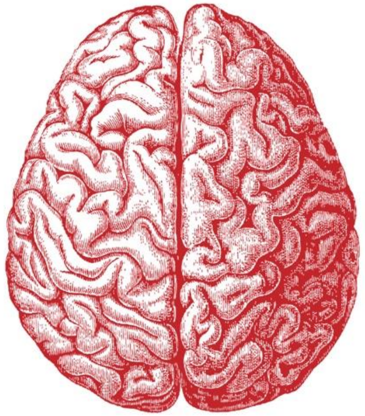
Why Draw Inspiration from Evolution?

2. Evolutionary Computing *(Inspired by Biological Evolution)*

Modeled intelligence using principles from **natural selection** and **evolution**.

Focused on the idea of evolving solutions over time through **mutation, crossover, and selection**.

Evolutionary computing became a powerful tool for solving **complex optimization problems**, leading to the development of **Genetic Algorithms (GAs)** and related techniques.



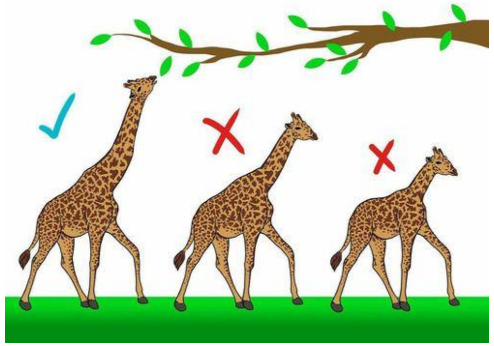
Motivation (2/2)

- **Intractability of Traditional Search Methods:**
  - Searching large search spaces with traditional methods is intractable.
  - This is especially true when states or candidate solutions have a large number of successors.
- Traditional search algorithms like brute force, exhaustive search, or gradient-based methods:
  - Require checking every possible state, which becomes computationally expensive
  - Often get stuck in local optima in complex search spaces.
  - **Result**: Inefficient and impractical for solving large, complex problems.

Evolution

- **Biological evolution:**
  - One of the mechanisms of evolution focuses mainly on **Natural Selection**,
  - **Natural selection** is the process by which certain traits become more or less common in a population over time due to their effect on the survival and reproduction of individuals.
  - Organisms with traits that provide a **fitness advantage** (better ability to survive and reproduce in their environment) are more likely to pass on their genes to the next generation.
  - Advantageous traits become more common, while disadvantageous traits may diminish or disappear.
  - **Charles Darwin** is credited with first extensively discussing and popularizing the idea of **natural selection** in *"On the Origin of Species"*, with **Alfred Russel Wallace** independently arriving at a similar theory around the same time.

Key Aspects of Natural Selection



- **Variation**: Individuals in a population have variations in traits (e.g., size, color, speed).
- **Heritability**: These traits are inherited from one generation to the next.
- **Differential Survival and Reproduction**: Some individuals with certain traits are more likely to survive and reproduce than others, leading to the "selection" of those traits.
- **Adaptation**: Over time, populations become better adapted to their environment as favorable traits become more common.

Biological Backgrounds 1/2

- **Basic Genetic Structure : DNA, Chromosomes, and Genes: The Foundation of Heredity**
  - DNA is the hereditary material in all living species.
  - Each cell contains **chromosomes** made up of DNA.
    - Chromosomes are **strings of DNA** ,
    - Echa Chromosome contains a set of genes (blocks of DNA) – carrying instructions for various traits.

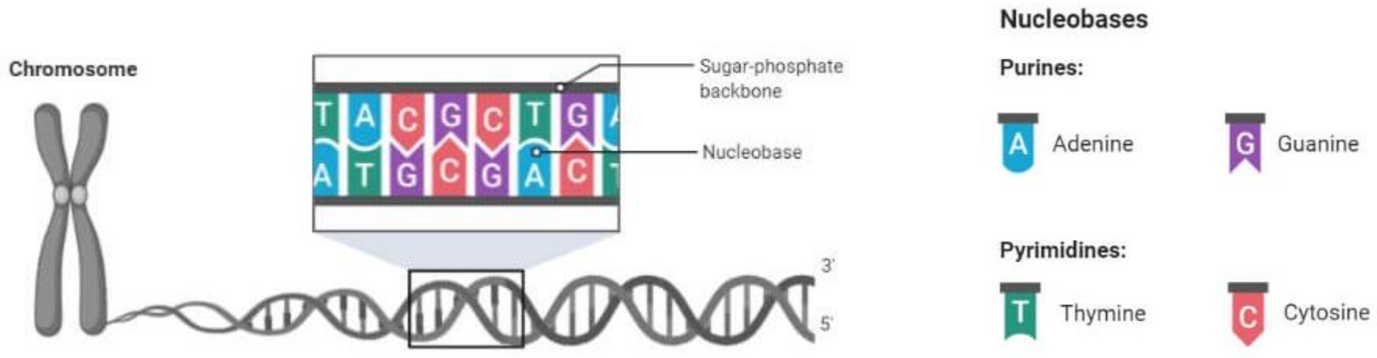


Image source

This content is for your knowledge and understanding only and will not be included in the exam

Biological Backgrounds 2/2

- **Genes** are sections of DNA that encode particular **traits**. (e.g., height, wing length, eye colour, and hair colour).
- Each gene has specific variations known as **alleles** (e.g., blue or brown eye color).
- The **locus** is the unique position of the gene on the chromosome.

- **From Genotype to Phenotype: Fitness and Reproduction**
  - **genome** is complete set of genetic materials
  - The **genotype** is the complete set of genes an organism carries.
  - The **phenotype** is the observable traits (e.g., eye color, height).
  - **Reproduction** involves the recombination of genes from parents
  - The **fitness** of an organism reflects how well it reproduces before it dies.

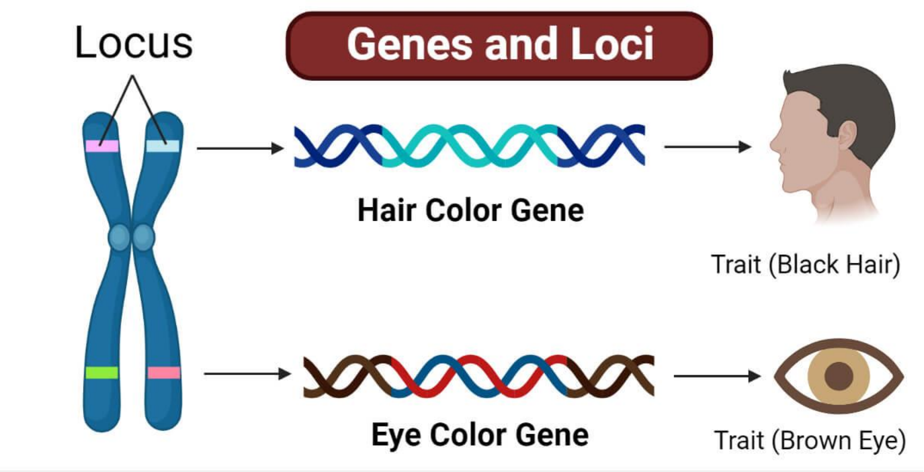


Image source

This content is for your knowledge and understanding only and will not be included in the exam

Evolutionary Algorithms (EAs)

- Developed by John Holland, University of Michigan (1970's)
- EAs fall into the category of "generate and test" algorithms
- They are stochastic, population-based algorithms
- **Variation** operators (recombination and mutation) create the necessary diversity and thereby facilitate novelty
- **Selection** reduces diversity and acts as a force pushing quality

Example: **Aircraft Wing Design**

Boeing used genetic algorithms to optimize the wing design of the 777 aircraft.

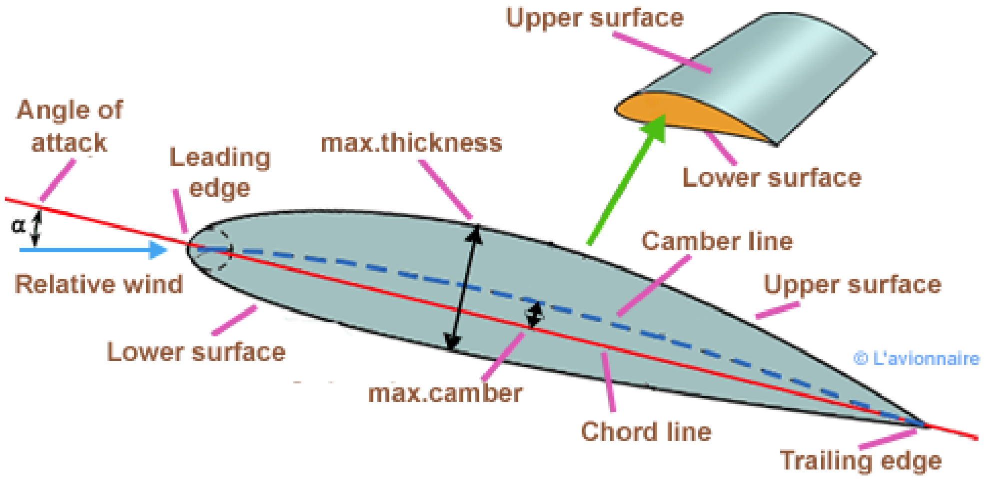
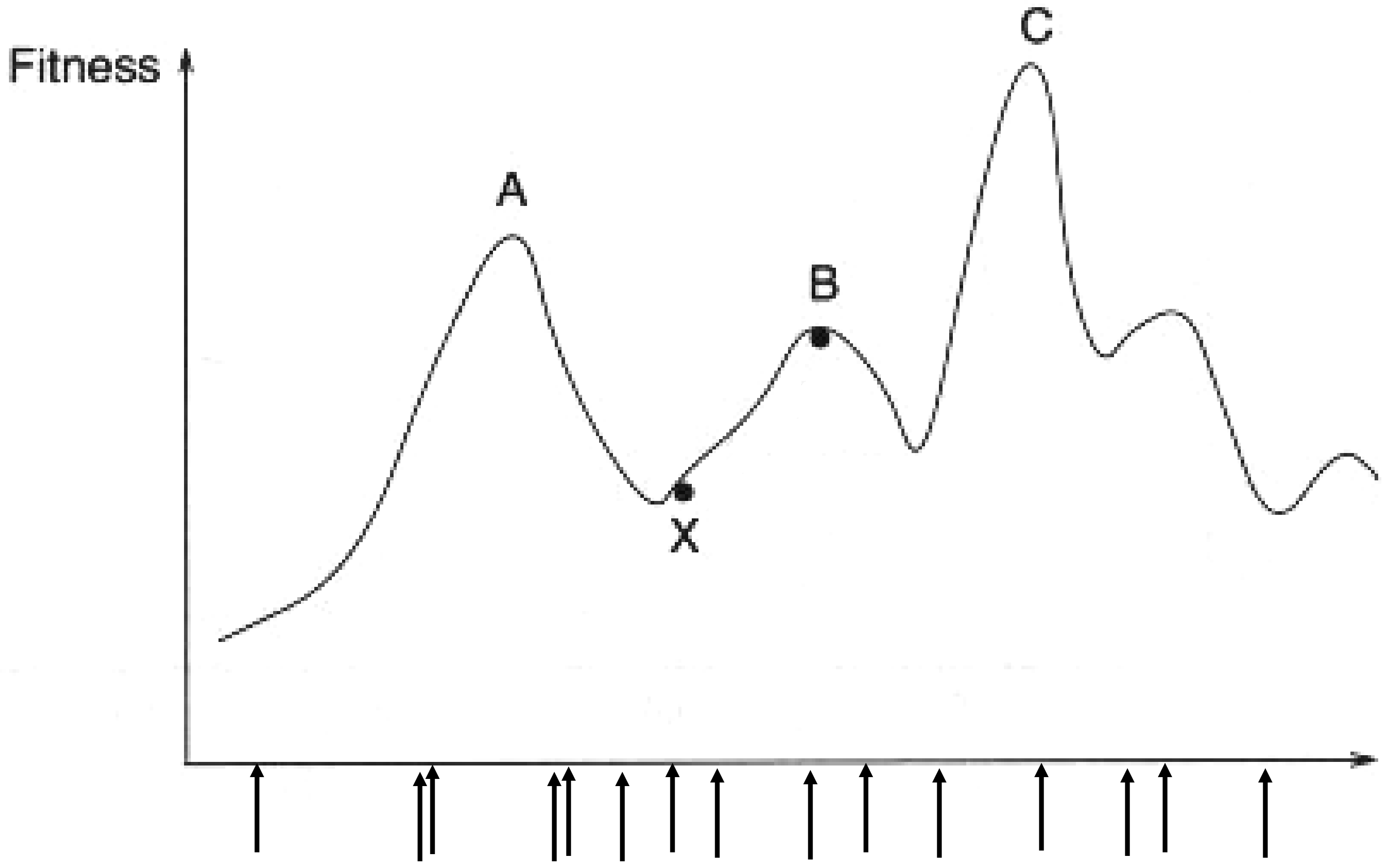
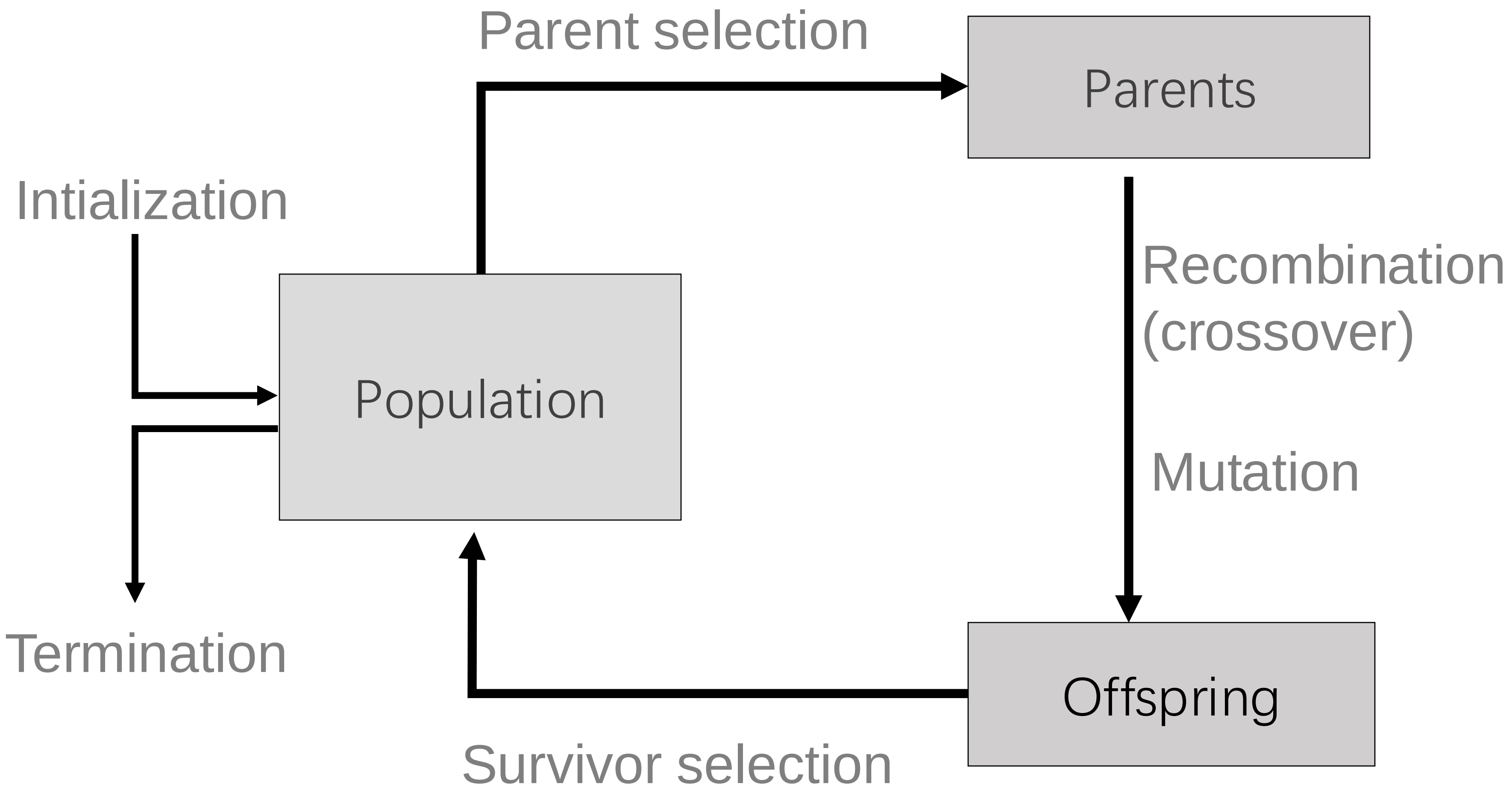


Image source

The Problem with Hillclimbing



General scheme of EAs



EA scheme in pseudo-code

```
BEGIN
  INITIALISE population with random candidate solutions;
  EVALUATE each candidate;
  REPEAT UNTIL ( TERMINATION CONDITION is satisfied ) DO
    1 SELECT parents;
    2 RECOMBINE pairs of parents;
    3 MUTATE the resulting offspring;
    4 EVALUATE new candidates;
    5 SELECT individuals for the next generation;
  OD
END
```

Scheme of an EA: Two pillars of evolution

There are two competing forces

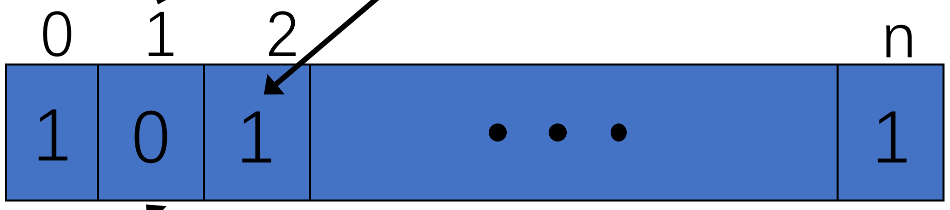
Increasing population **diversity** by genetic operators • mutation • recombination Push towards **novelty**	**Decreasing** population **diversity** by selection • of parents • of survivors Push towards **quality**

Exploration/
Exploitation

Representation: EA terms

Locus: the position of a gene

Allele= 0 or 1 (what values a gene can have)



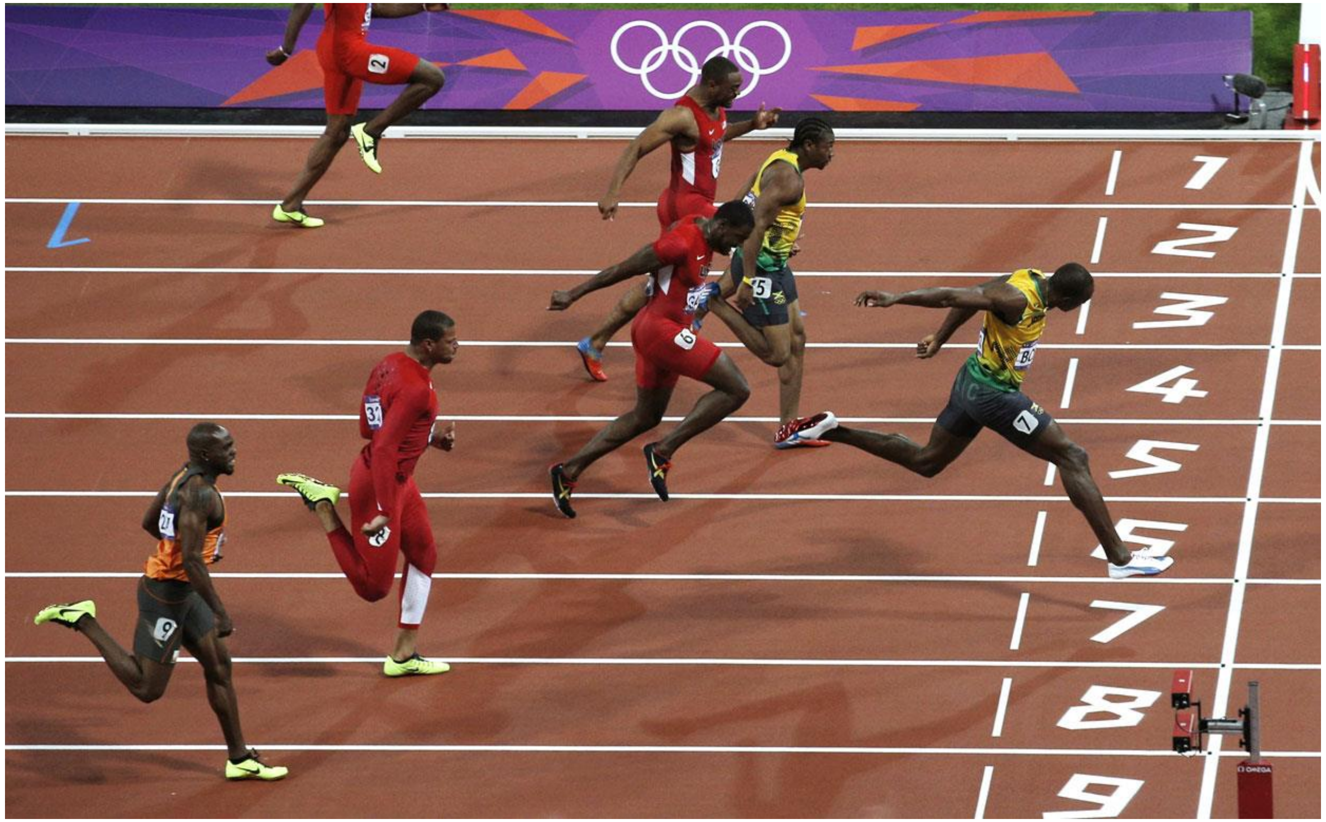
Gene: one element of the array

Genotype: a set of gene values

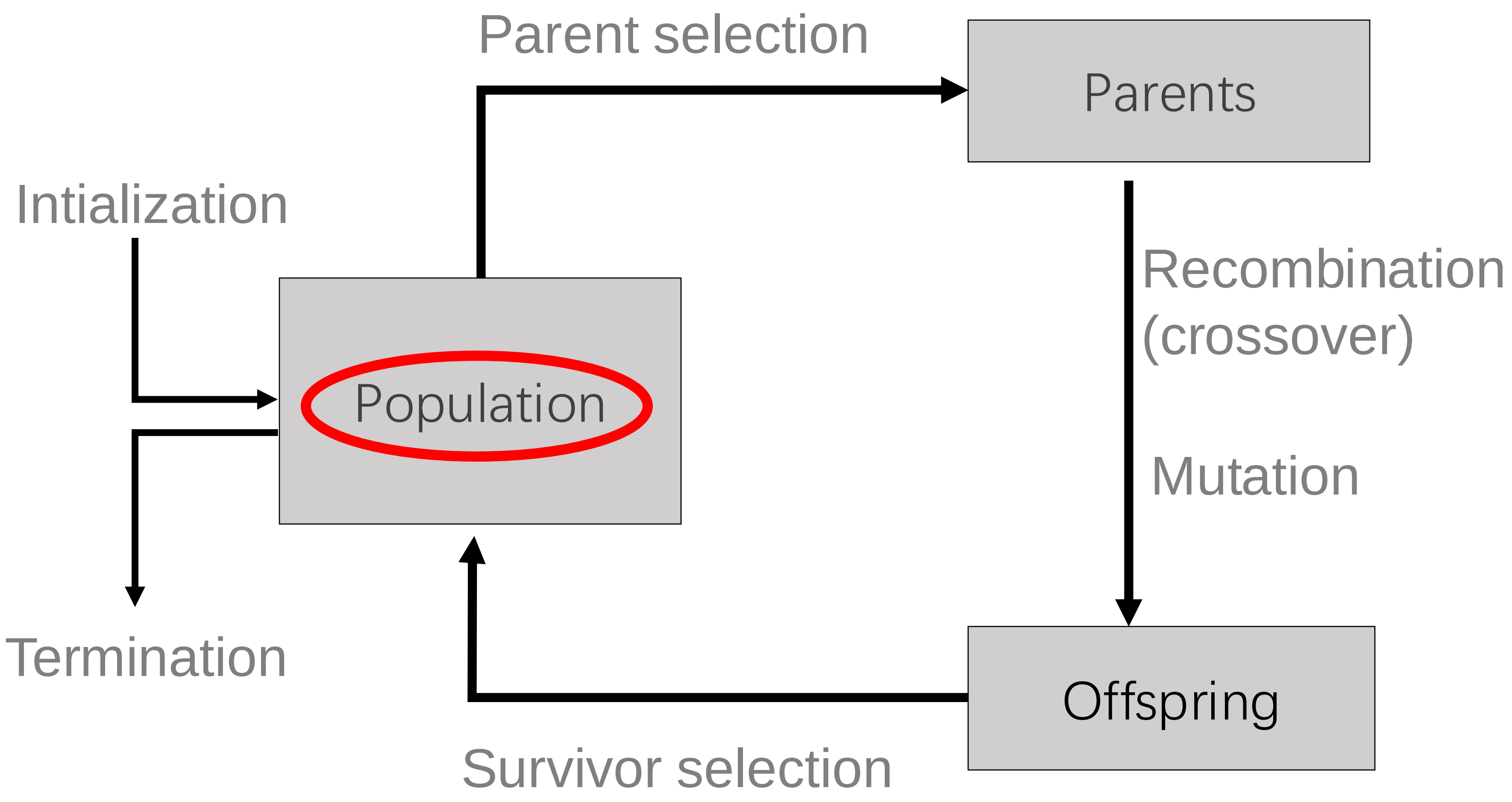
Phenotype: what could be built/developed based on the genotype

Main EA components: Evaluation (fitness) function

- Represents the task to solve
- Enables selection (provides basis for comparison)
- Assigns a single real-valued fitness to each phenotype



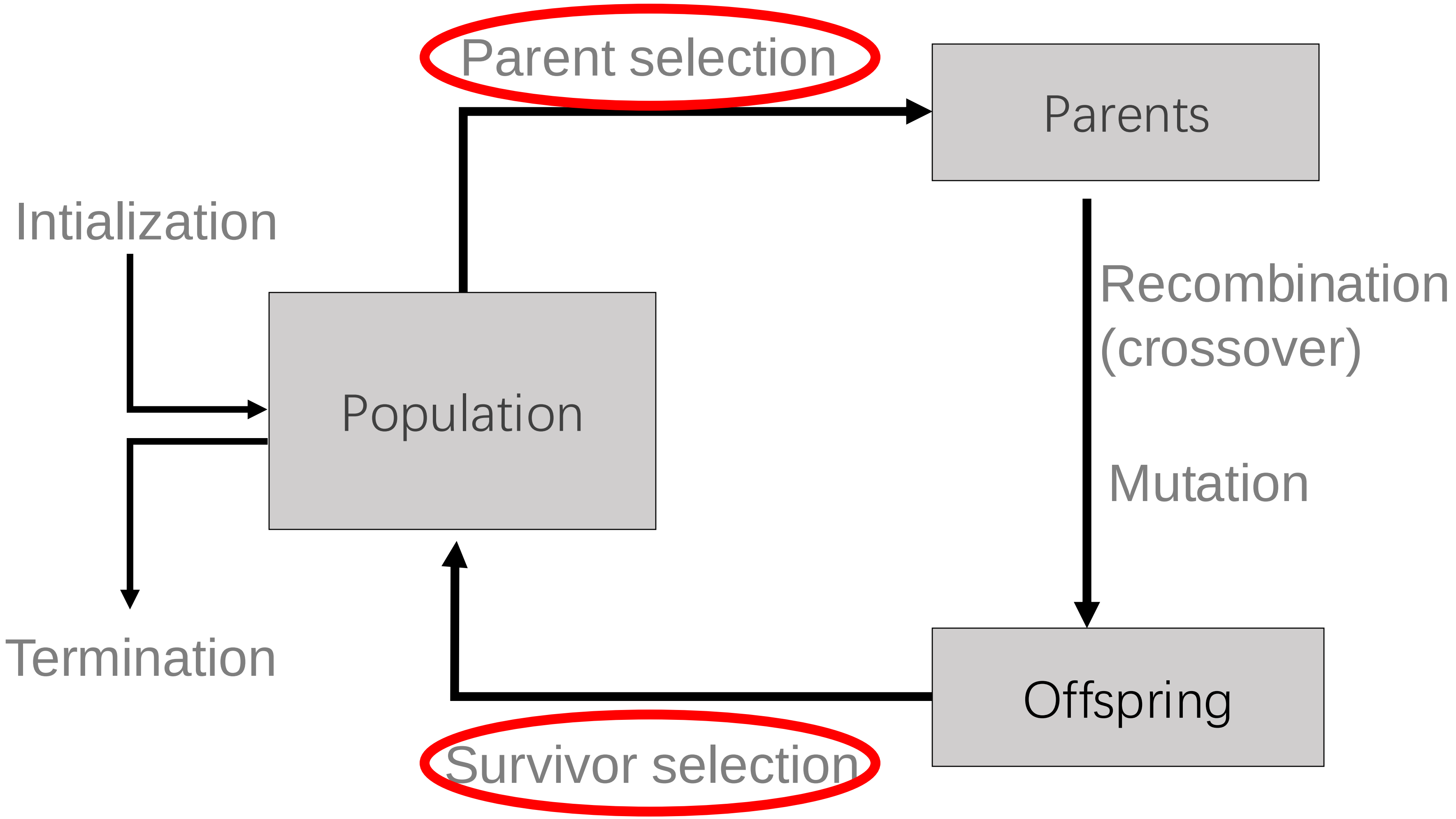
General scheme of EAs



Main EA components: Population

- The **candidate solutions (individuals)** of the problem
- Population is the basic unit of evolution, i.e., the **population is evolving**, not the individuals
- **Selection** operators act on **population level**
- **Variation** operators act on **individual level**

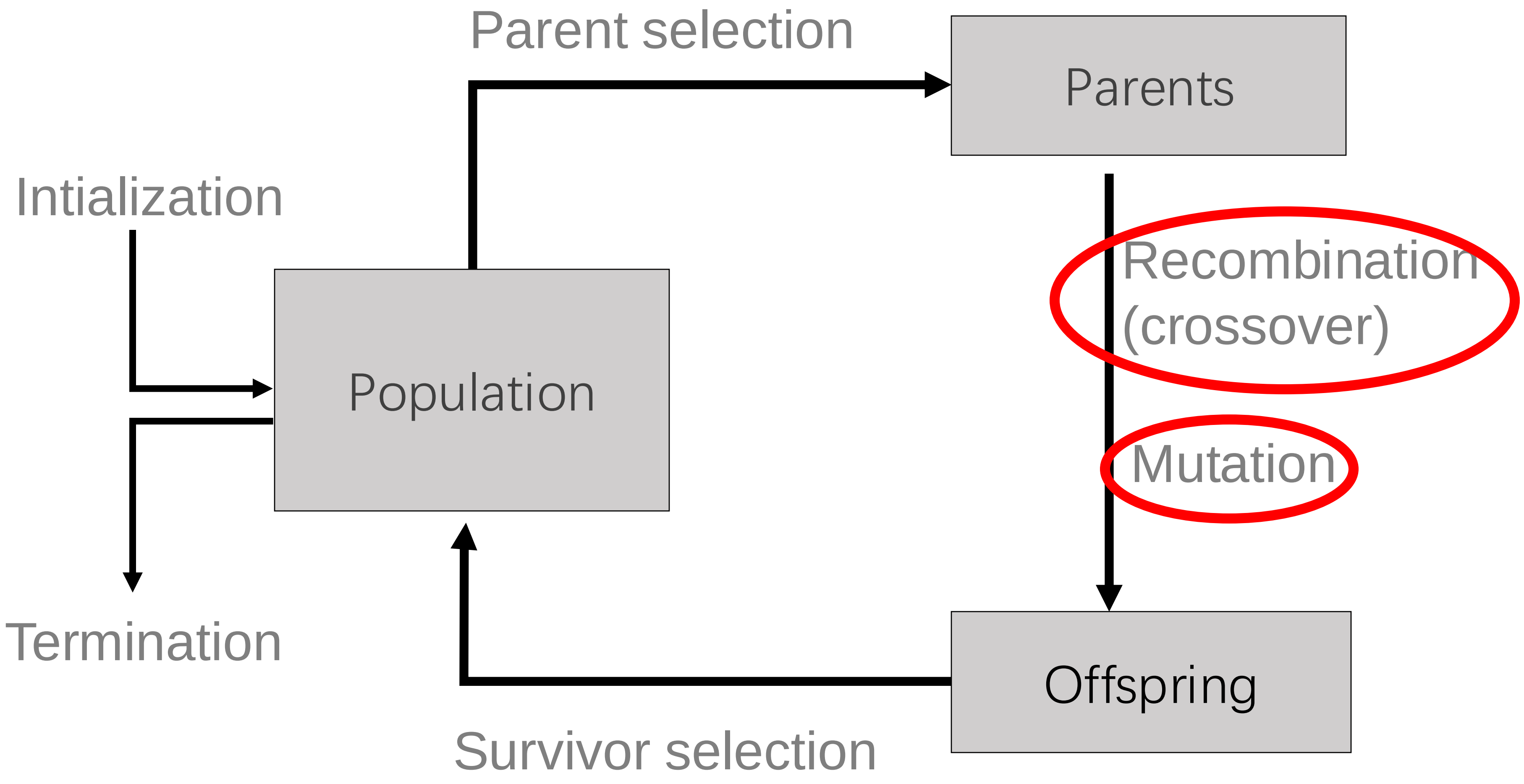
General scheme of EAs



Main EA components: Selection mechanisms

- Identify individuals
  - to become parents
  - to survive
- Pushes population towards higher fitness
- Parent selection is usually probabilistic
  - high quality solutions more likely to be selected than low quality, but not guaranteed
  - This *stochastic* nature can aid escape from local optima
- More on selection next week!

General scheme of EAs

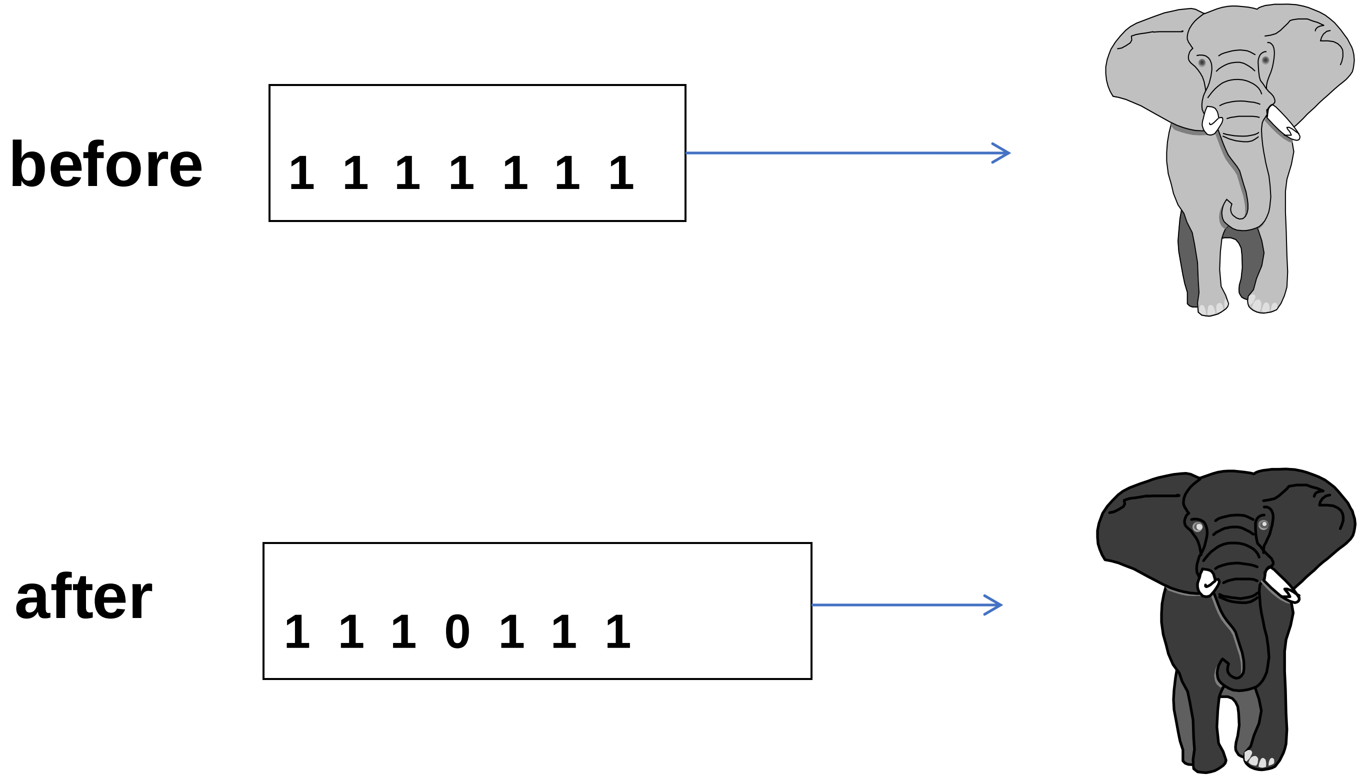


Main EA components: Variation operators

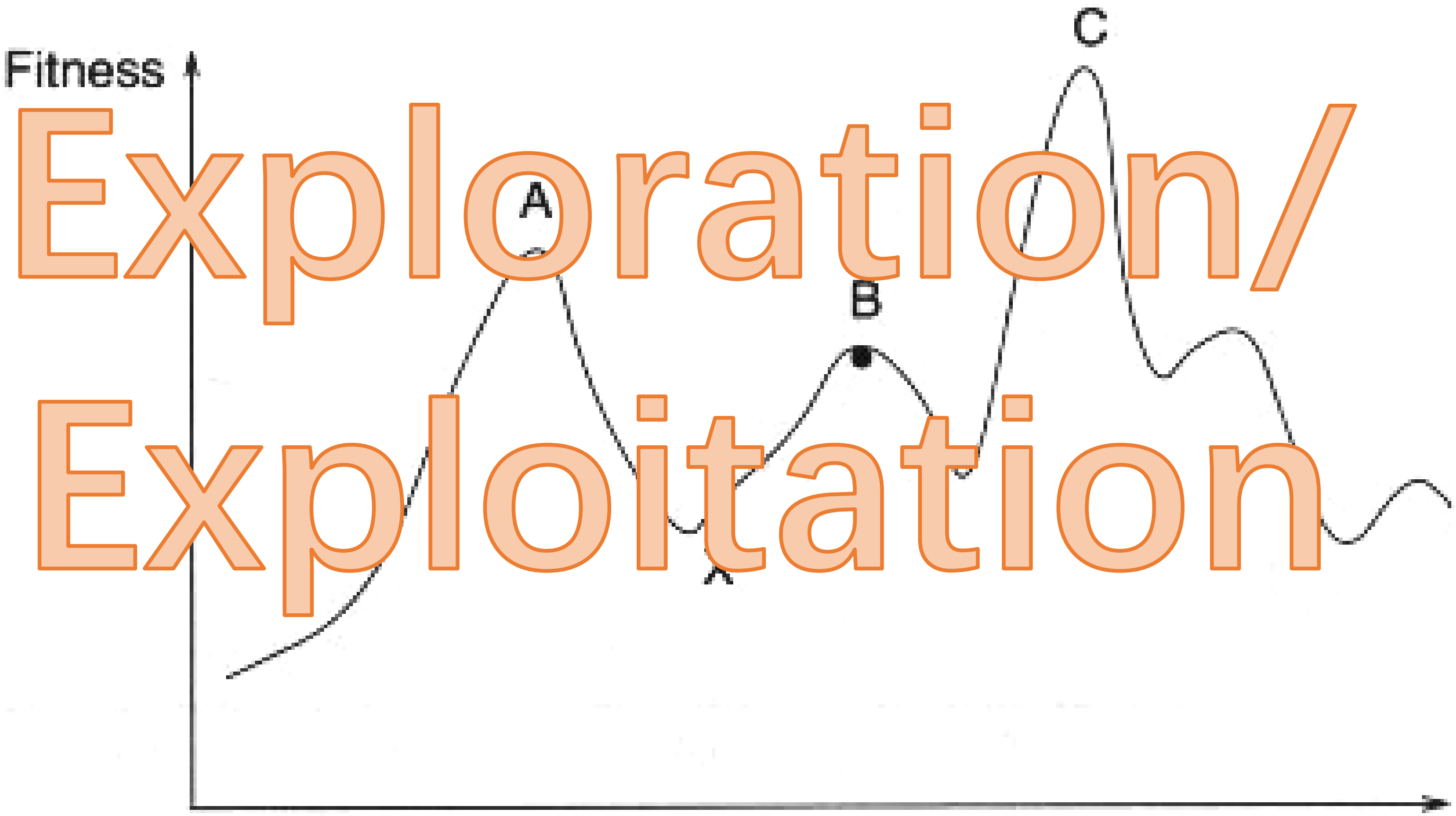
- Role: to generate new candidate solutions
- Usually divided into two types according to their **arity** (number of inputs to the variation operator):
  - Arity 1 : **mutation** operators
  - Arity >1 : **recombination** operators
  - Arity = 2 typically called **crossover**
  - Arity > 2 is formally possible, seldom used in EC

Main EA components: Mutation

- Role: cause small, random variance to a genotype
- Element of **randomness is essential** and differentiates it from other unary heuristic operators



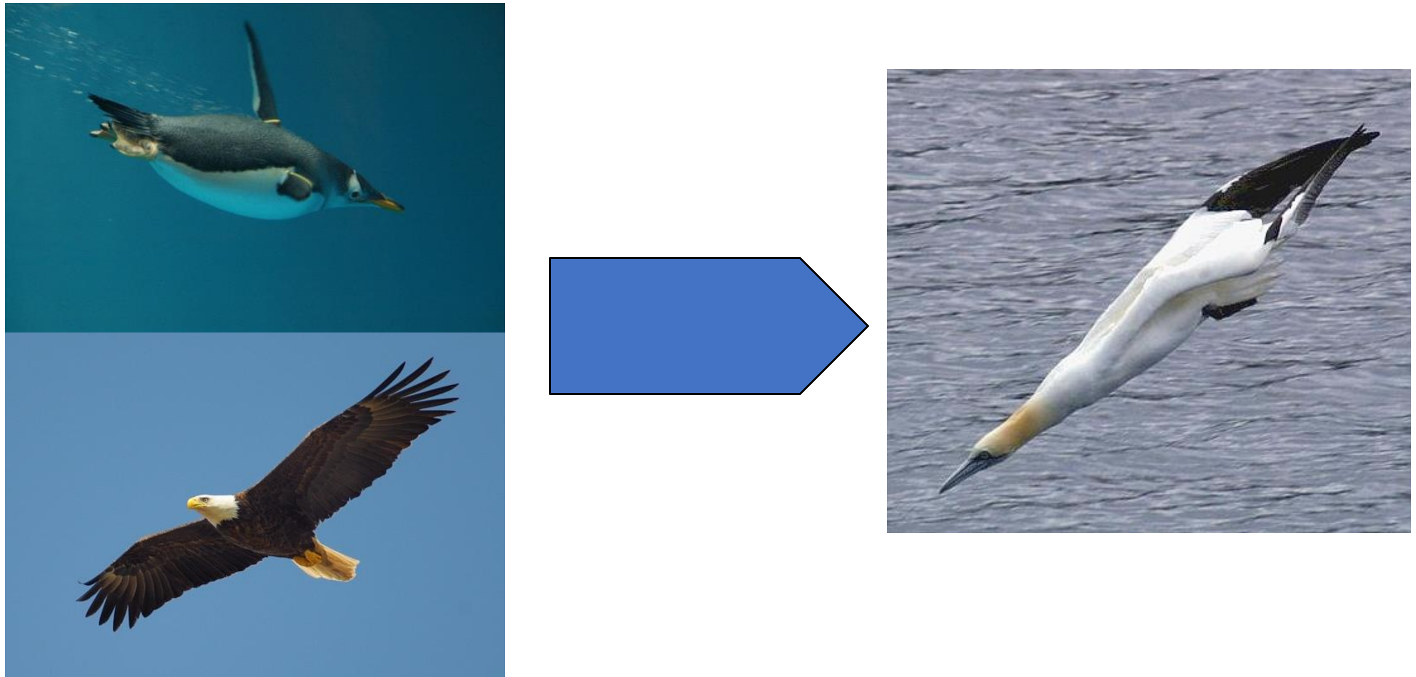
Why do we do Random Mutation?



Exploration/ Exploitation

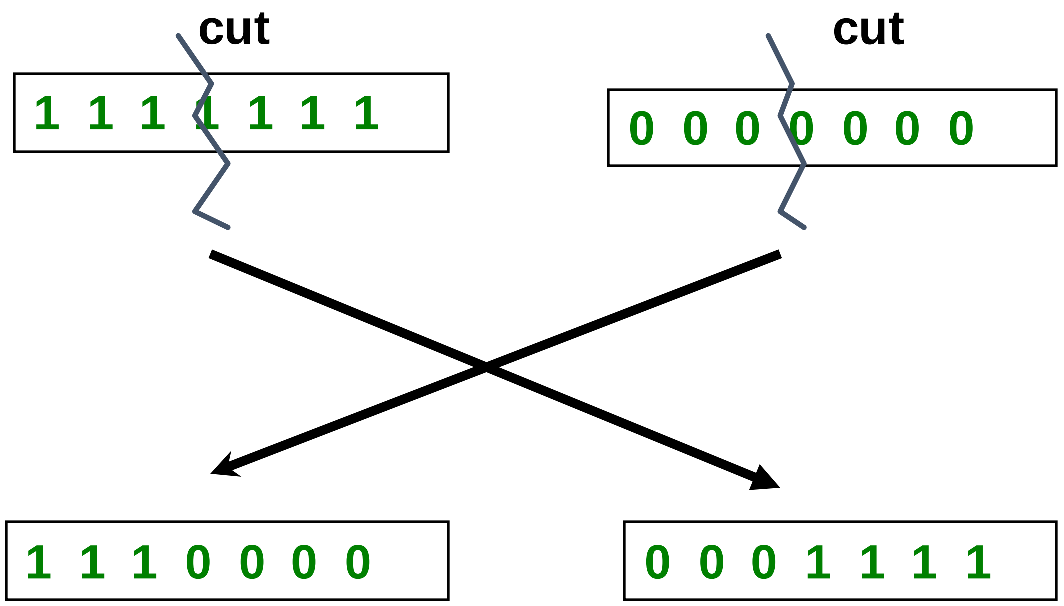
Main EA components: Recombination (1/3)

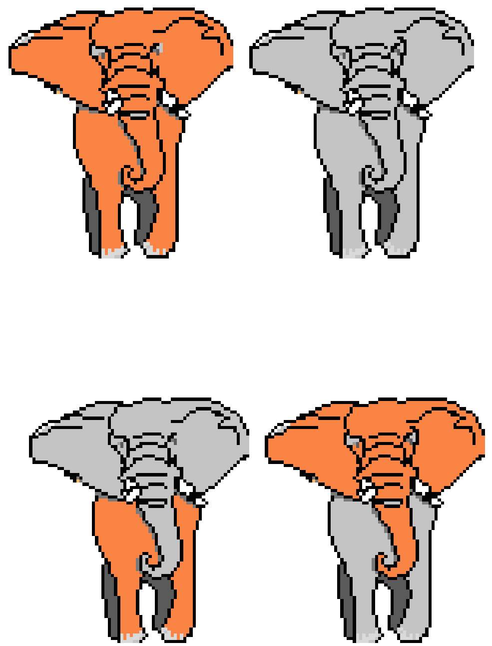
- Role: merges information from parents into offspring
- Choice of what information to merge is stochastic
- Hope is that some offspring are better by combining elements of genotypes that lead to good traits



Main EA components: Recombination (2/3)

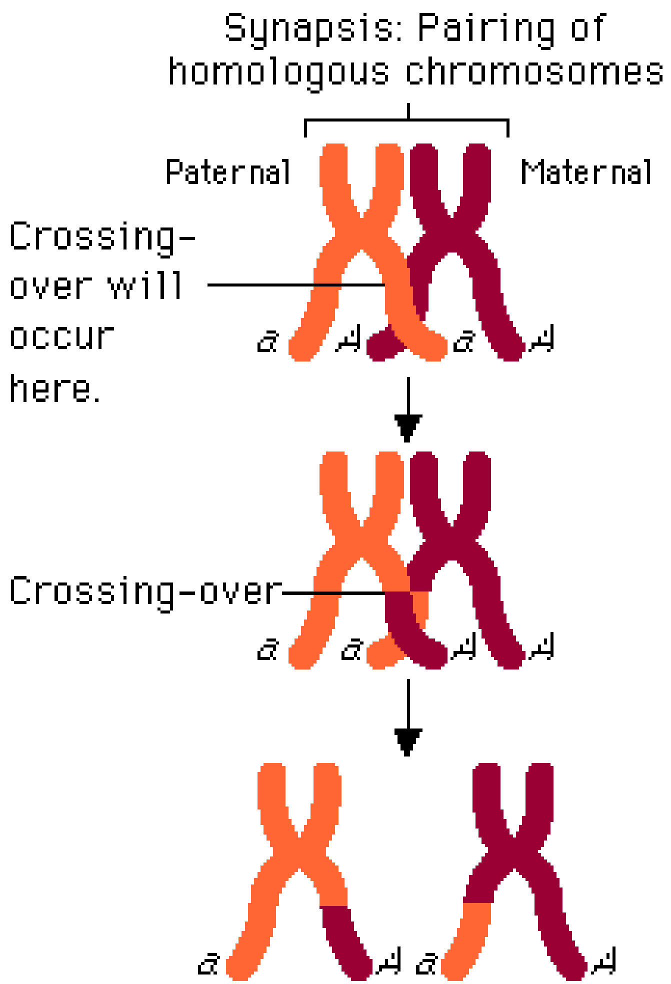
Parents





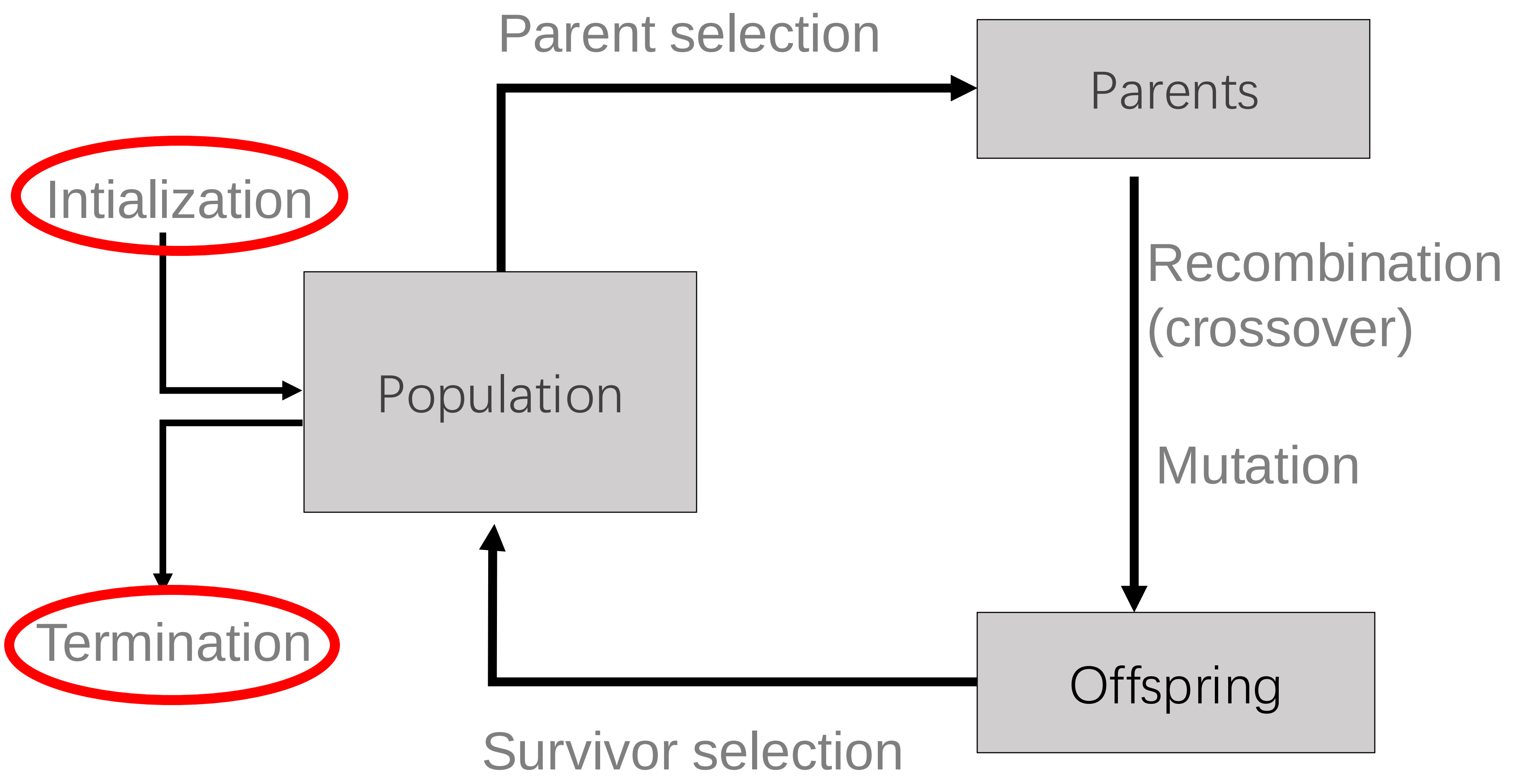
Offspring

Main EA components: Recombination/cross-over in nature (3/3)



This content is for your knowledge and understanding only and will not be included in the exam

General scheme of EAs



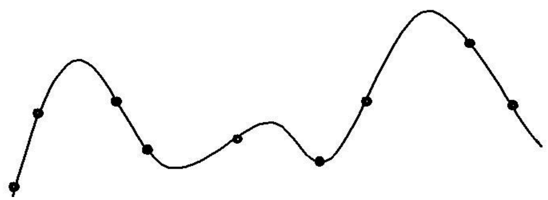
Main EA components: Initialisation / Termination

- Initialisation usually done at random,
  - Need to ensure even spread and mixture of possible allele values
  - Can include existing solutions, or use problem-specific heuristics, to “seed” the population

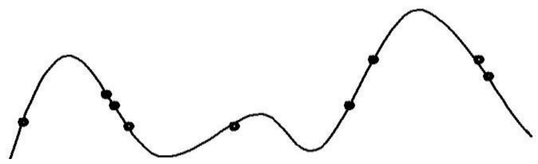
- Termination condition checked every generation
  - Reaching some (known/hoped for) fitness
  - Reaching some maximum allowed number of generations
  - Reaching some minimum level of diversity
  - Reaching some specified number of generations without fitness improvement

Typical EA behaviour: Stages

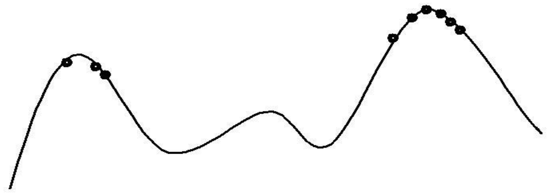
Stages in optimising on a 1-dimensional fitness landscape



Early stage:
quasi-random population distribution



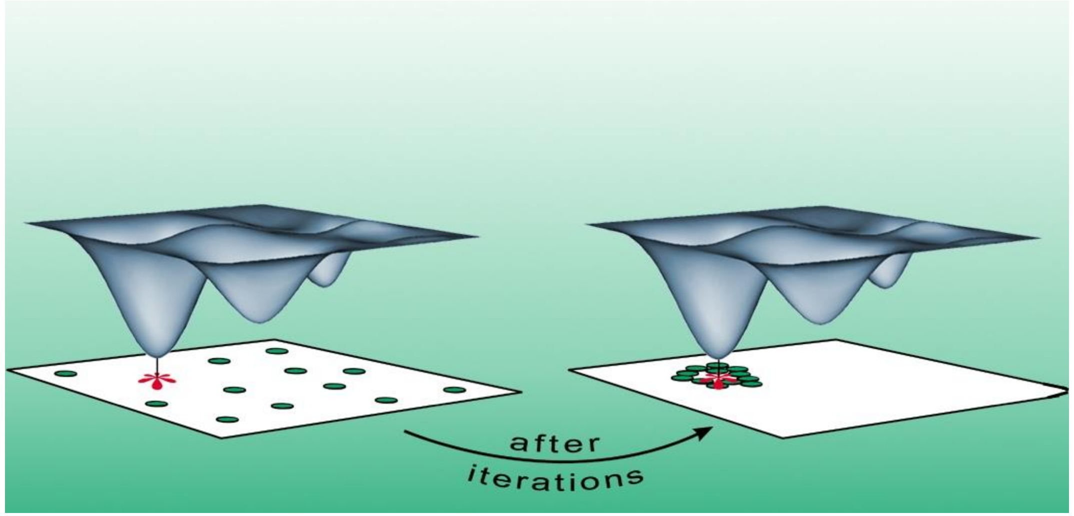
Mid-stage:
population arranged around/on hills



Late stage:
population concentrated on high hills

Applicable situations for GAs

- Optimal Performance of GA with a Population

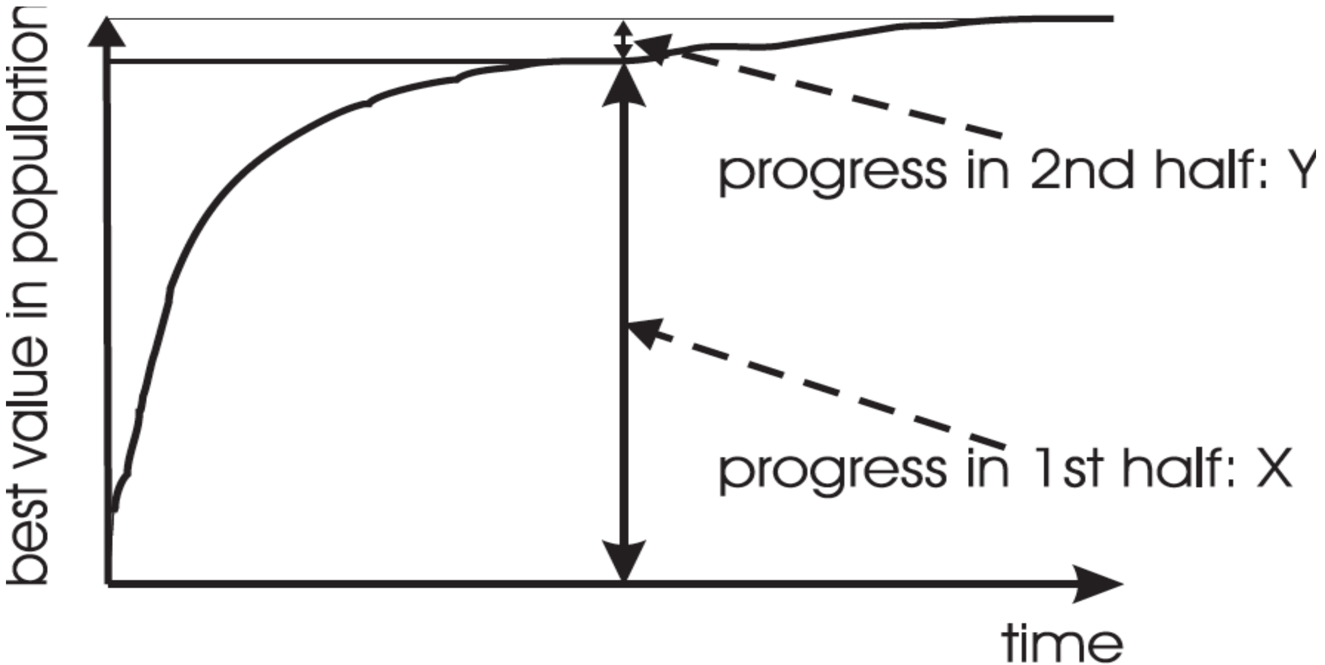


- Genetic Algorithms (GA) are generally effective in **discrete optimization** problems, with a **continuous fitness landscape**
  - In simple terms, this means we can use a heuristic to estimate how close a candidate is to becoming a solution.
  - ❖ *Good problems for GA*: e.g. knapsack problem
  - ❖ *Bad problems for GA*: e.g. Finding large primes (why?)

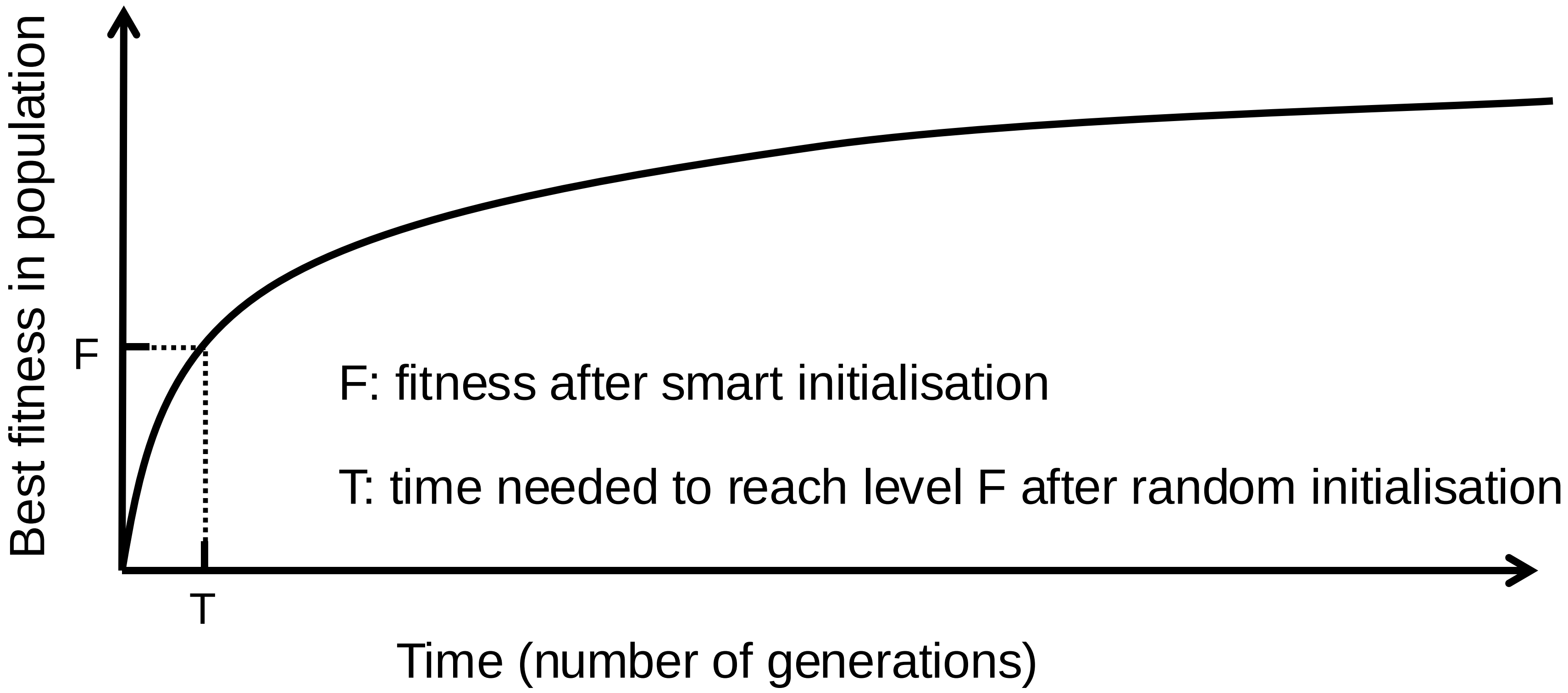
Typical EA behaviour: Typical run: progression of fitness

Typical EA behaviour: Are long runs beneficial?

- Answer:
  - It depends on how much you want the last bit of progress

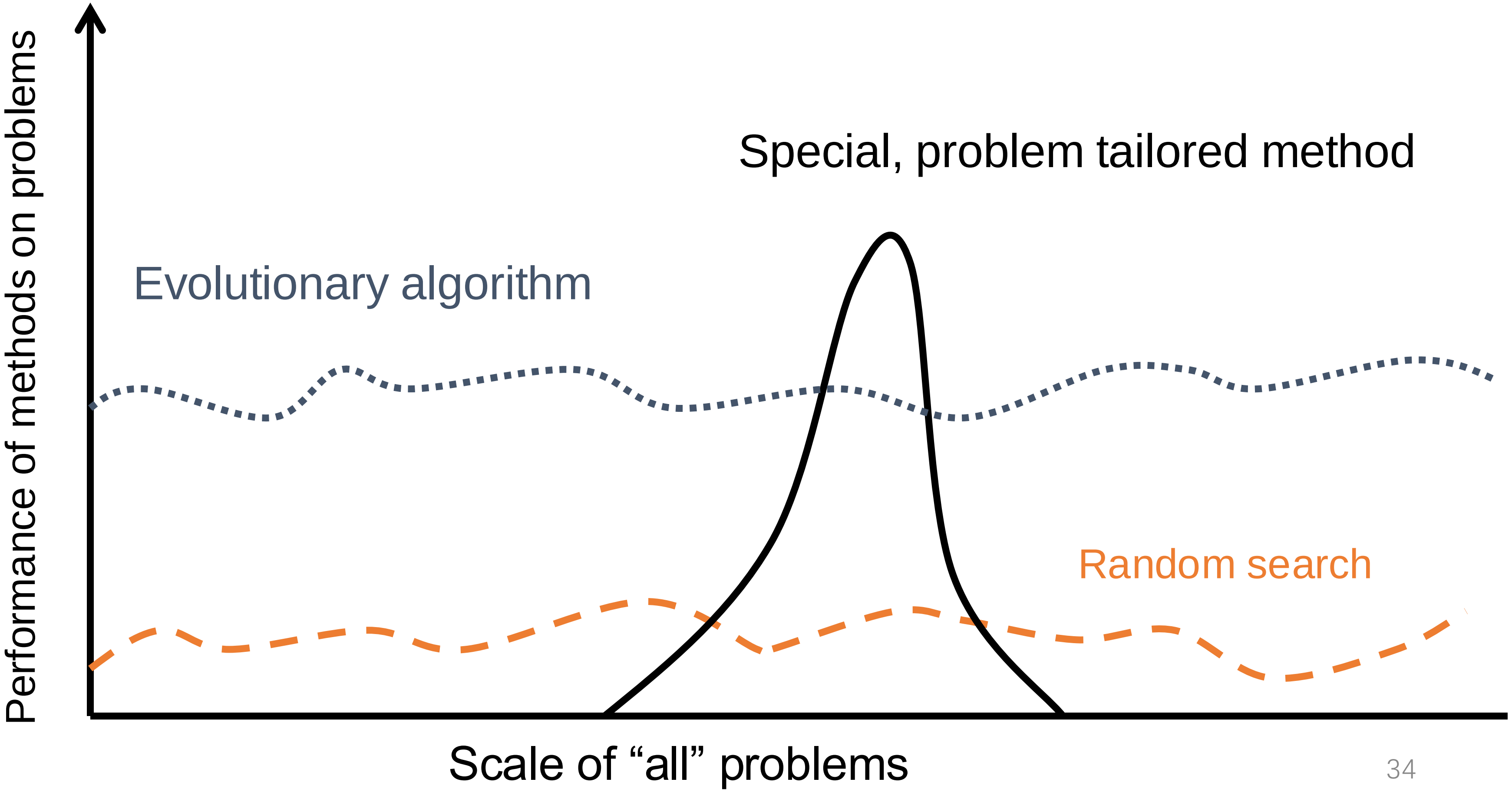


Typical EA behaviour: Is it worth expending effort on smart initialisation?



- Answer: it depends.
  - Possibly good, if good solutions/methods exist.
  - Care is needed, see chapter/lecture on hybridisation.

Traditional View on EA Performance



Typical EA behaviour: EAs and domain knowledge

- Trend in the 90's:

  adding problem specific knowledge to EAs

  (special variation operators, repair, etc)

- Result: EA performance curve "deformation":
  - better on problems of the given type
  - worse on problems different from given type
  - amount of added knowledge is variable

- Recent theory suggests the search for an "all-purpose" algorithm may be fruitless

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

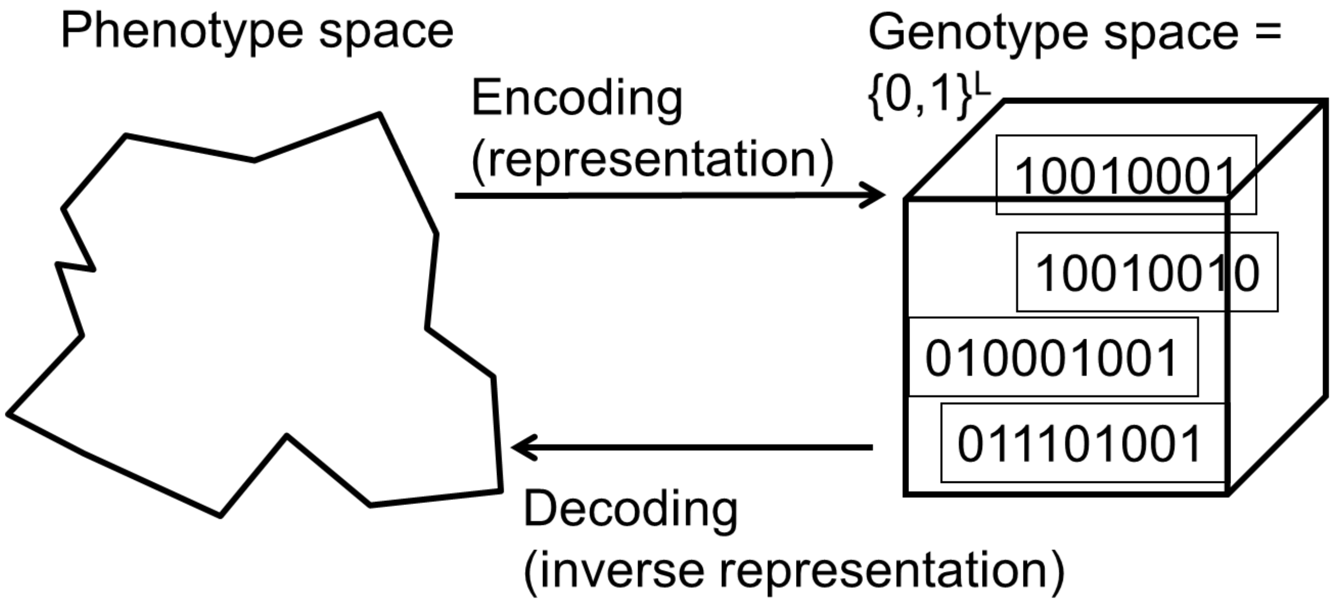
Role of representation and variation operators

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

TSP: How to represent?

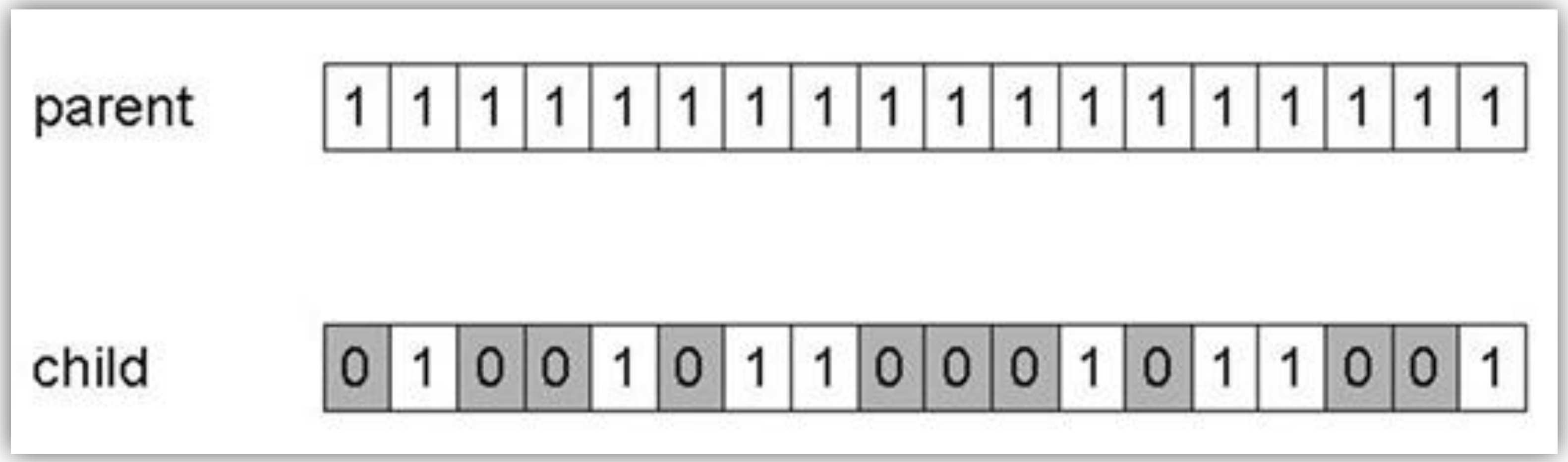
Binary Representation

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



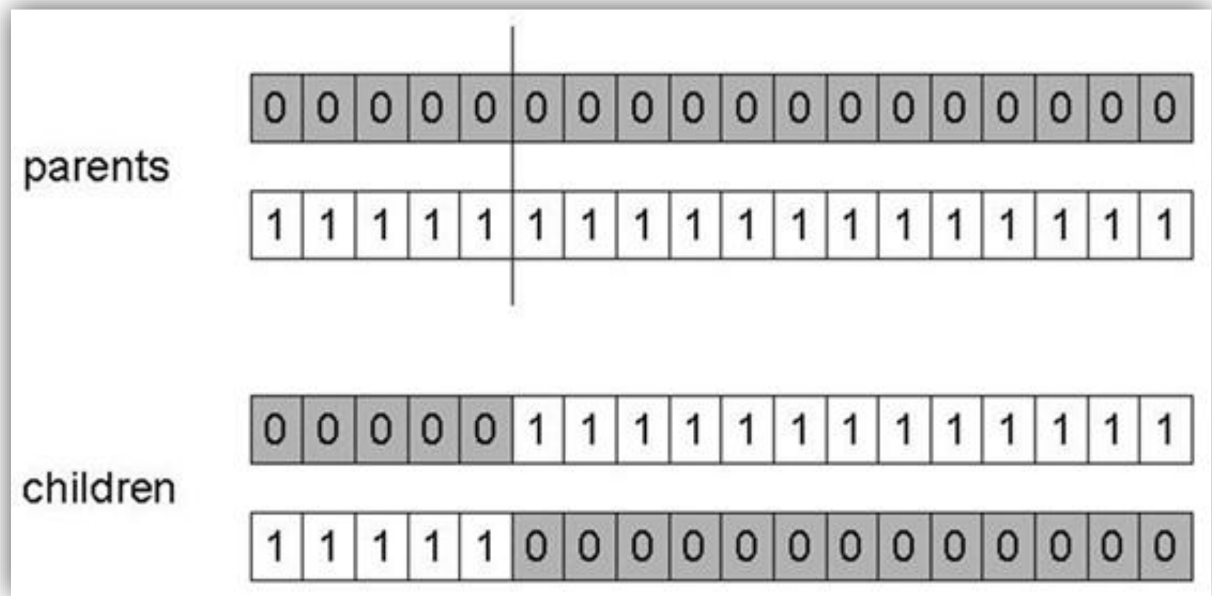
Binary Representation: Mutation

- Alter each gene independently with a probability p_m
- p_m is called the mutation rate



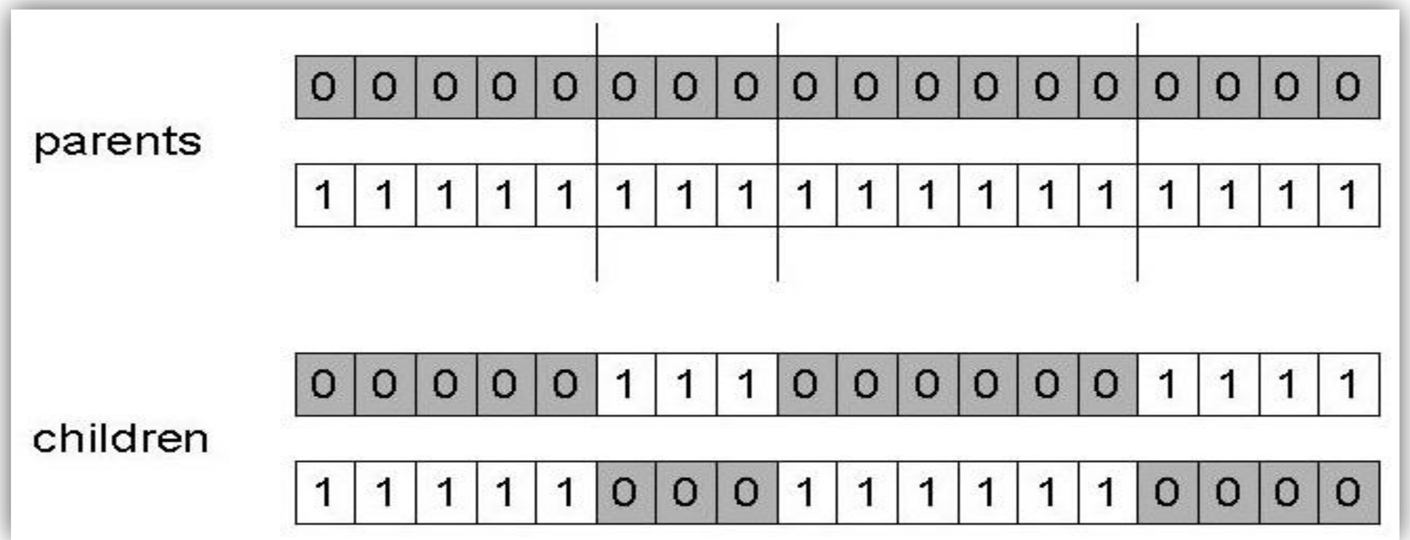
Binary Representation: 1-point crossover

- Choose a random point on the two parents
- Split parents at this crossover point
- Create children by exchanging tails



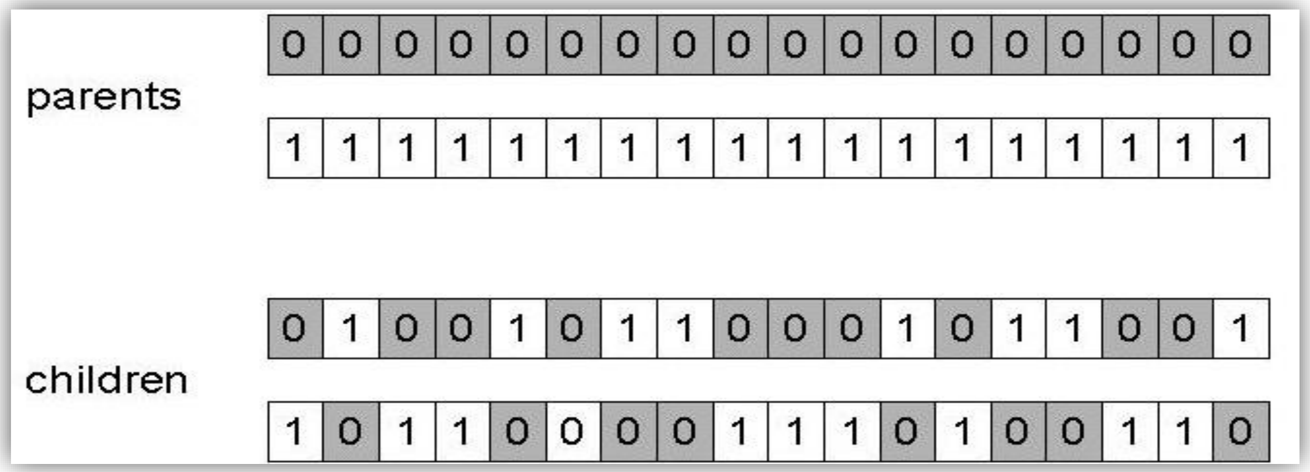
Binary Representation: n-point crossover

- Choose n random crossover points
- Split along those points
- Glue parts, alternating between parents



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
- Breaks more “links” in the genome



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

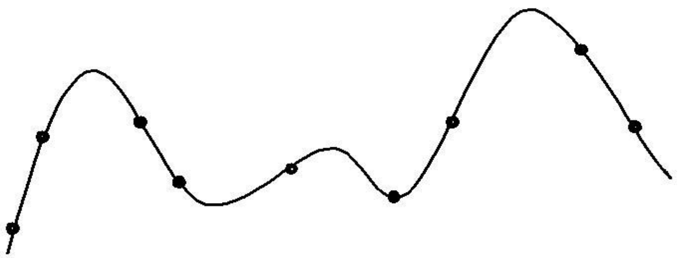
- Decade long debate:
  - which one is better / necessary ?

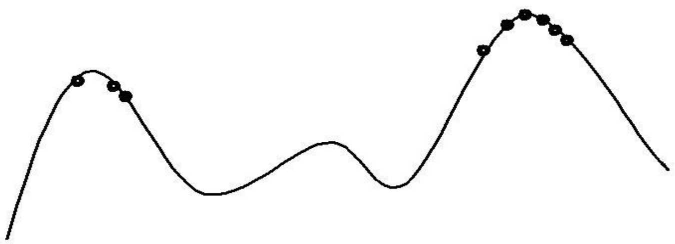
- Answer (at least, rather wide agreement):
  - it depends on the problem, but in general, it is good to have both
  - both have a different role
  - mutation-only-EA is possible, x-over-only-EA would not work

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

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

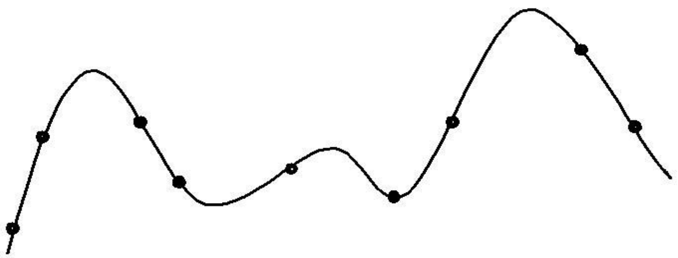
Exploitation: Optimising within a promising area, i.e. using information

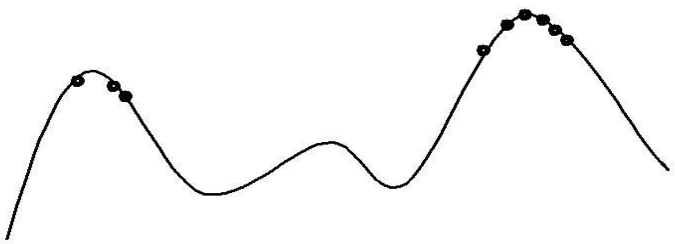




Crossover and/or mutation?

- **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





Genetic Algorithms: An example after Goldberg '89

- Simple problem: max x^2 over $\{0,1,\cdots,31\}$
- GA approach:
  - Representation: binary code, e.g., 01101 $\leftrightarrow$ 13
  - Population size: 4
  - 1-point x-over, bitwise mutation
  - Roulette wheel selection
  - Random initialisation
- We show one generational cycle done by hand

X^2 example: Parent Selection

String no.	Initial population	x Value	Fitness $f(x) = x^2$	$Prob_i$	Expected count	Actual count
1	0 1 1 0 1	13	169	0.14	0.58	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Sum			1170	1.00	4.00	4
Average			293	0.25	1.00	1
Max			576	0.49	1.97	2

X^2 example: Crossover

String no.	Mating pool	Crossover point	Offspring after xover	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 | 1	4	0 1 1 0 0	12	144
2	1 1 0 0 | 0	4	1 1 0 0 1	25	625
2	1 1 | 0 0 0	2	1 1 0 1 1	27	729
4	1 0 | 0 1 1	2	1 0 0 0 0	16	256
Sum					1754
Average					439
Max					729

X^2 example: Mutation

String no.	Offspring after xover	Offspring after mutation	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 0	$\boxed{1}$ 1 1 0 0	26	676
2	1 1 0 0 1	1 1 0 0 1	25	625
2	1 1 0 1 1	1 1 0 1 1	27	729
4	1 0 0 0 0	1 0 $\boxed{1}$ 0 0	18	324
Sum				2354
Average				588.5
Max				729

Integer Representation

- 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

Real-Valued or Floating-Point Representation: Uniform Mutation

- General scheme of floating point mutations

$$\bar{x} = \langle x_1, \ ..., x_l \rangle \rightarrow \bar{x}' = \langle x'_1, ..., x'_l \rangle$$

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

- **Uniform Mutation:** x_i¢ drawn randomly (uniform) from $[LB_i, UB_i]$

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

Real-Valued or Floating-Point Representation: Nonuniform Mutation

- Non-uniform mutations:
  - 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 (- σ to + σ))

Real-Valued or Floating-Point Representation: Crossover operators

- Discrete recombination:
  - 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**

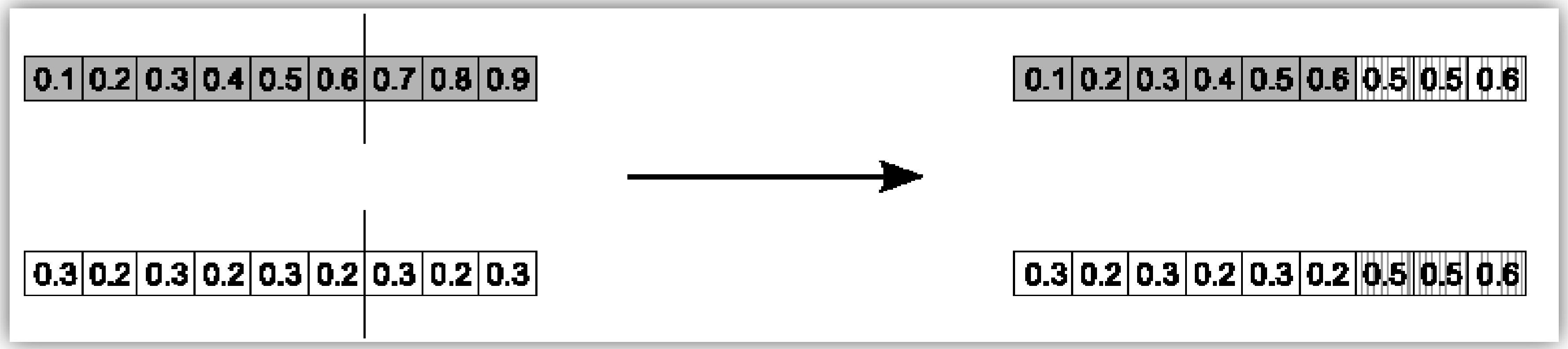
- Intermediate recombination:
  - exploits idea of creating children "between" parents (hence a.k.a. *arithmetic* recombination)
  - $z_i = \alpha x_i + (1 - \alpha) y_i$ where $\alpha : 0 \leq \alpha \leq 1$.
  - The parameter α can be:
    - constant: $\alpha = 0.5$ -> uniform arithmetical crossover
    - variable (e.g. depend on the age of the population)
    - picked at random every time

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

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

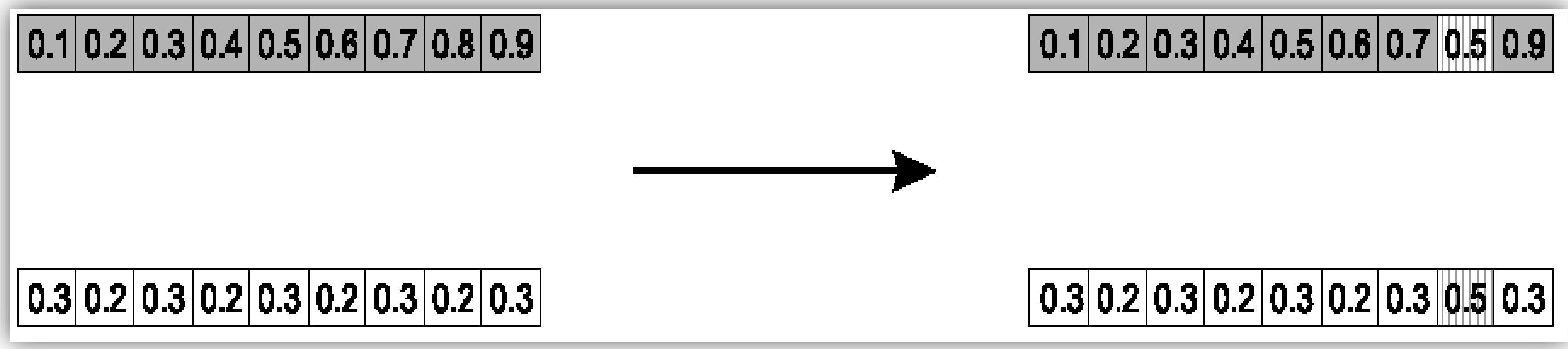
$$\left\langle x_1, \ldots, x_k, \alpha \cdot y_{k+1} + (1-\alpha) \cdot x_{k+1}, \ldots, \alpha \cdot y_n + (1-\alpha) \cdot x_n \right\rangle$$

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



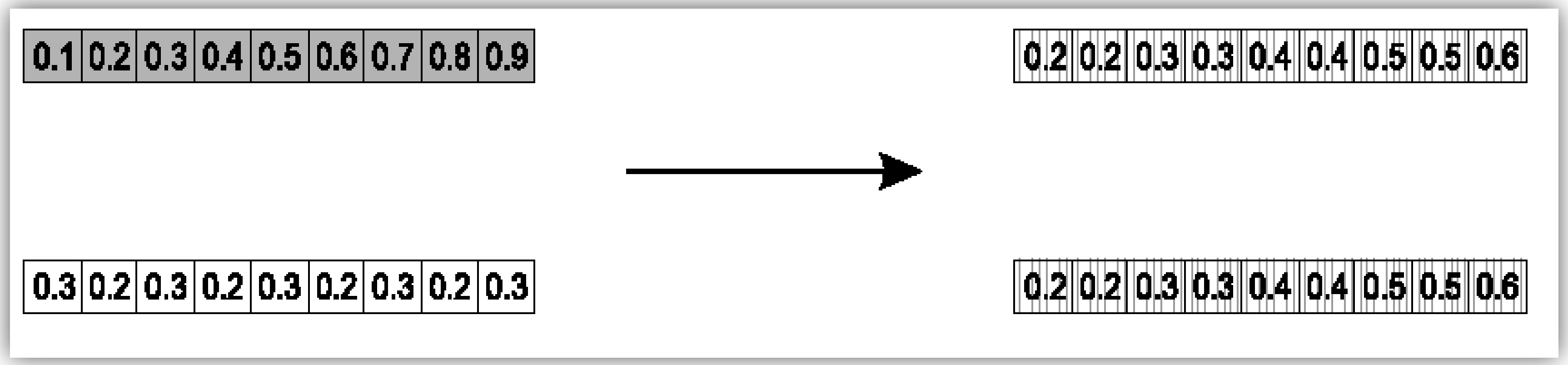
Real-Valued or Floating-Point Representation: Single arithmetic crossover

- Parents: $\langle x_1,\cdots,x_n \rangle$ and $\langle y_1,\cdots,y_n \rangle$
- Pick a single gene (k) at random,
- child$_1$ is: $\left\langle x_1, ..., x_k, \alpha \cdot y_k + (1-\alpha) \cdot x_k, ..., x_n \right\rangle$
- Reverse for other child. e.g. with $\alpha = 0.5$



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

- Most commonly used
- Parents: $\langle x_1,\cdots,x_n \rangle$ and $\langle y_1,\cdots,y_n \rangle$
- Child$_1$ is: $$a \cdot \bar{x} + (1-a) \cdot \bar{y}$$
- reverse for other child. e.g. with $\alpha = 0.5$



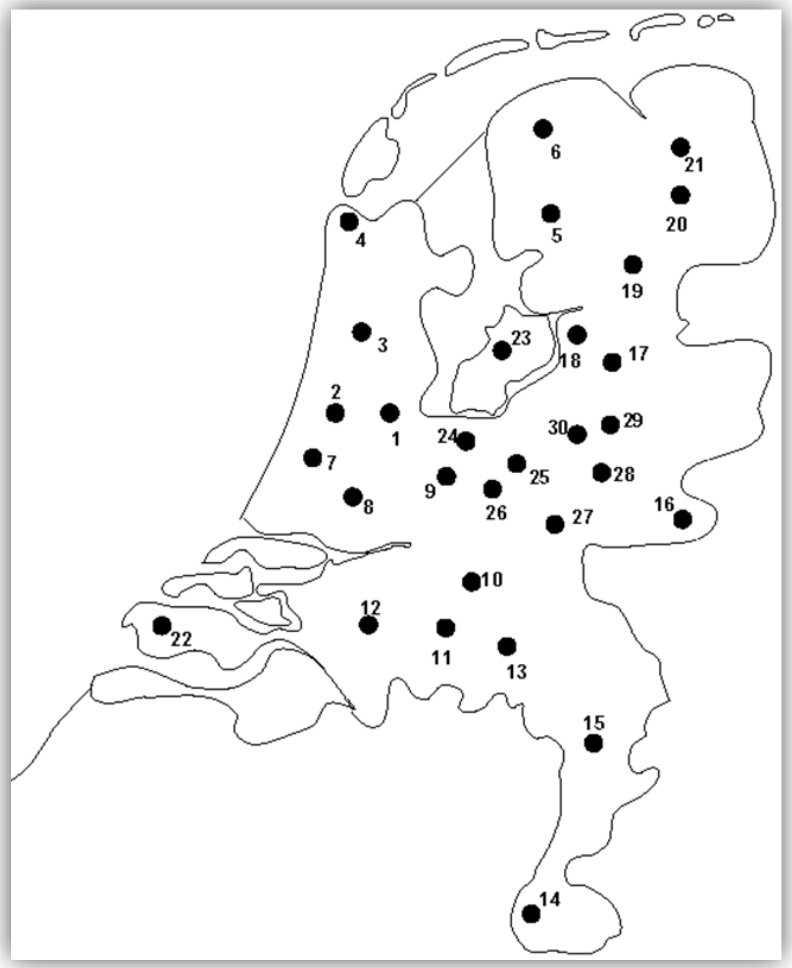
Permutation Representations

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

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 =5 and with origin city: 5, [1,2,3,4], [3,4,2,1] are OK)
- Search space is BIG:

  for 30 cities there are (30-1)! ≈ 10^{31} possible tours

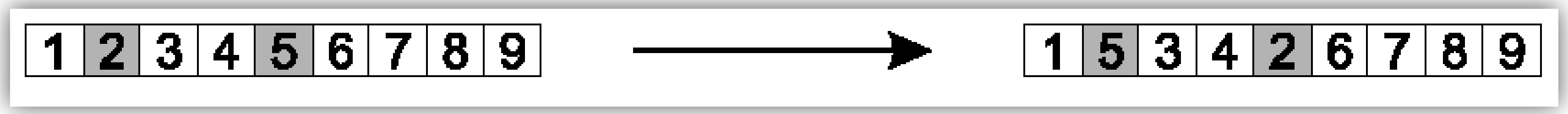


Permutation Representations: Mutation

- Normal mutation operators lead to inadmissible solutions
  - Mutating a single gene destroys the permutation
- Therefore, must change at least two values
- Mutation probability now reflects the probability that some operator is applied once to the whole string, rather than individually in each position

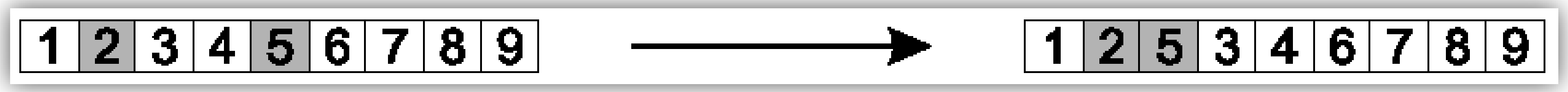
Permutation Representations: Swap mutation

- 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



Permutation Representations: Scramble mutation

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

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

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

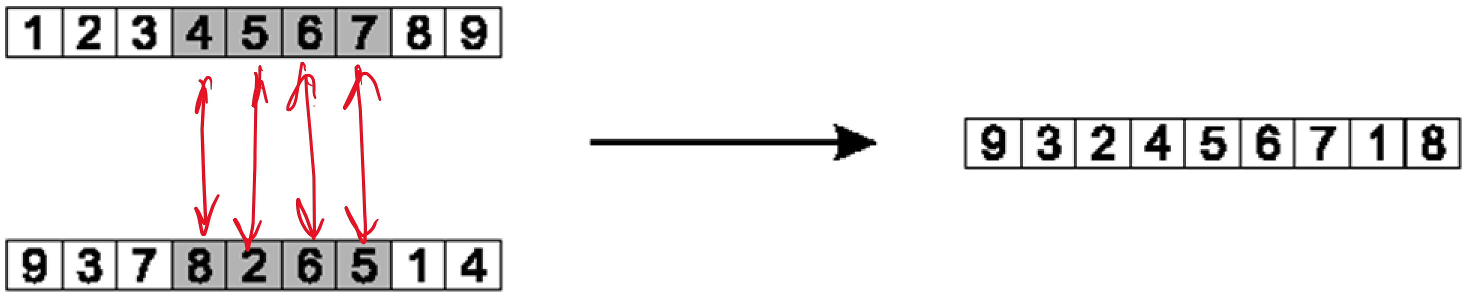
- Many specialised operators have been devised which focus on combining order or adjacency information from the two parents

Permutation Representations: Conserving Adjacency

- Important for problems where adjacency between elements decides quality (e.g. TSP)

Permutation Representations: Conserving Adjacency

- Important for problems where adjacency between elements decides quality (e.g. TSP)
  - [1,2,3,4,5] is same plan as [5,4,3,2,1] -> order and position not important, but adjacency is.
- **Partially Mapped Crossover** and Edge Recombination are example operators



Permutation Representations: Conserving Order

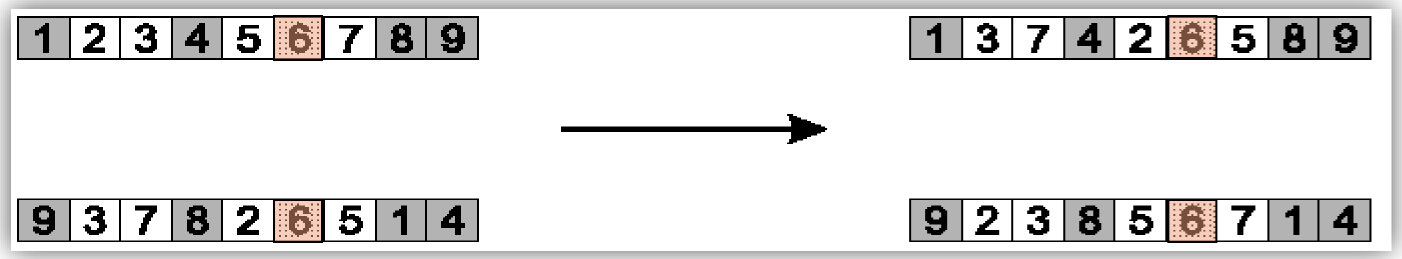
- Important for problems where **order** of elements decide performance (e.g. production scheduling)

Making breakfast:

1. **Start brewing coffee**
2. **Toast bread**
3. **Apply butter**
4. **Add jam**
5. **Pour hot coffee**

Permutation Representations: Conserving Order

- Important for problems order of elements decide performance (e.g. production scheduling)
  - Now, [1,2,3,4,5] is a very different plan than [5,4,3,2,1]
- Order Crossover and **Cycle Crossover** are example operators



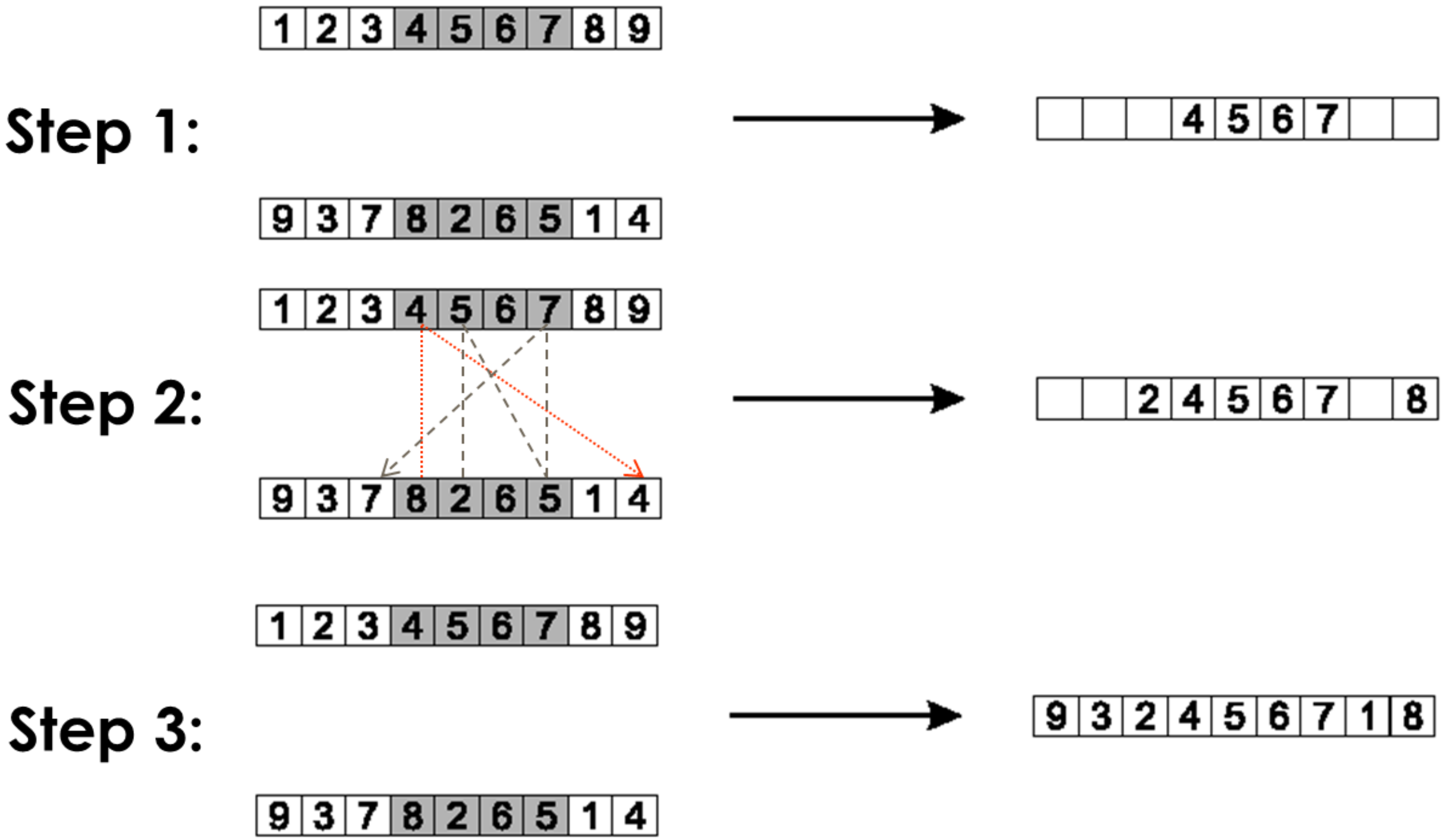
Permutation Representations: Partially Mapped Crossover (PMX) (1/2)

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

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

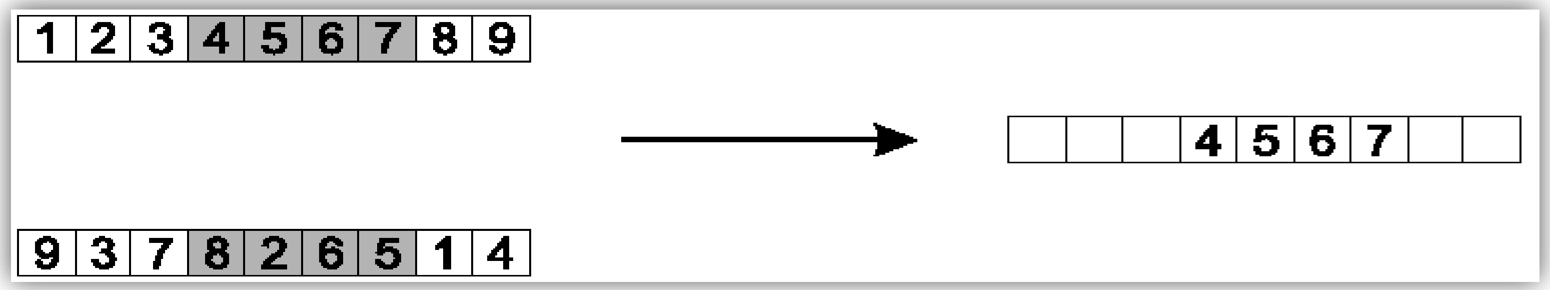


Permutation Representations: Order crossover (1/2)

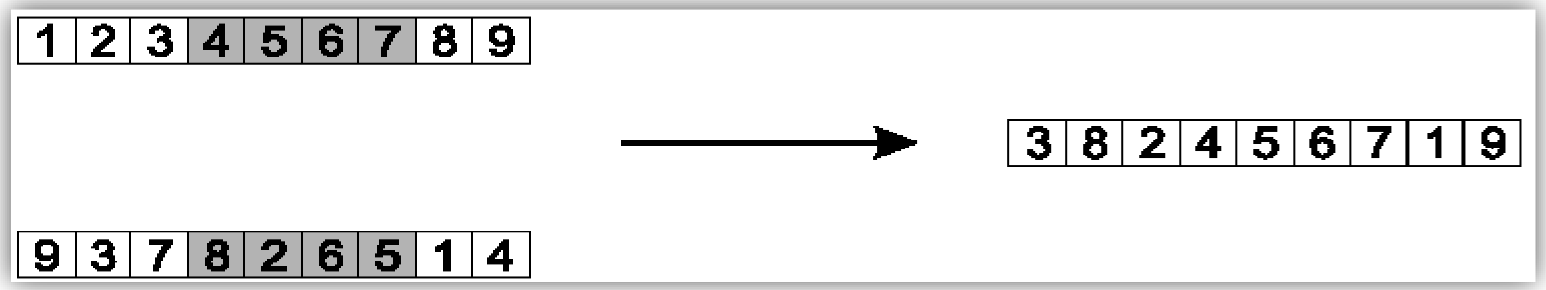
- Idea is to preserve relative order that elements occur
- Informal procedure:
  - 1. Choose an arbitrary part from the first parent
  - 2. Copy this part to the first child
  - 3. Copy the numbers that are not in the first part, to the first child:
    - starting right from cut point of the copied part,
    - 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 crossover (2/2)

- Copy randomly selected set from first parent



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



Permutation Representations: Cycle crossover (1/2)

Basic idea:

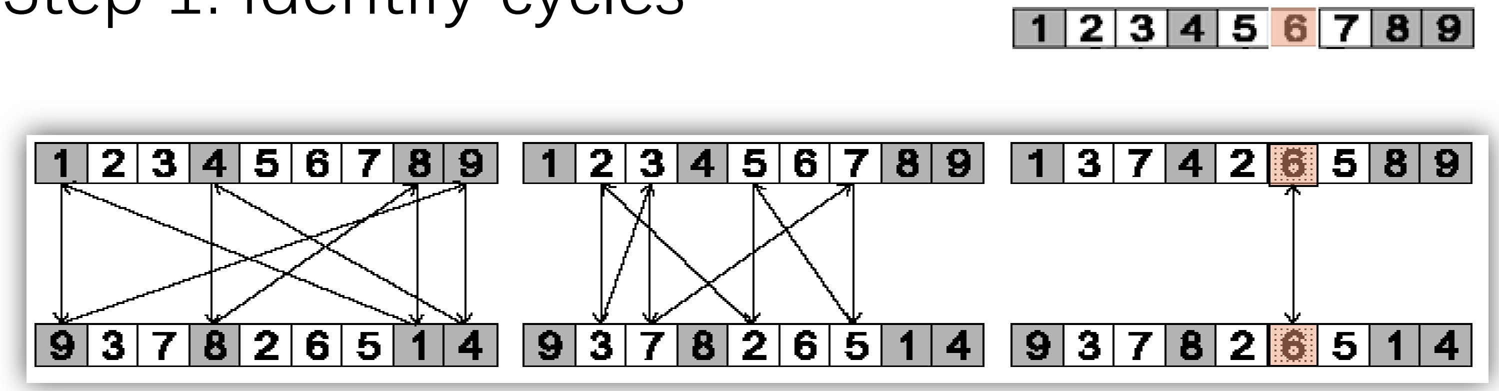
Each allele comes from one parent *together with its position*.

Informal procedure:

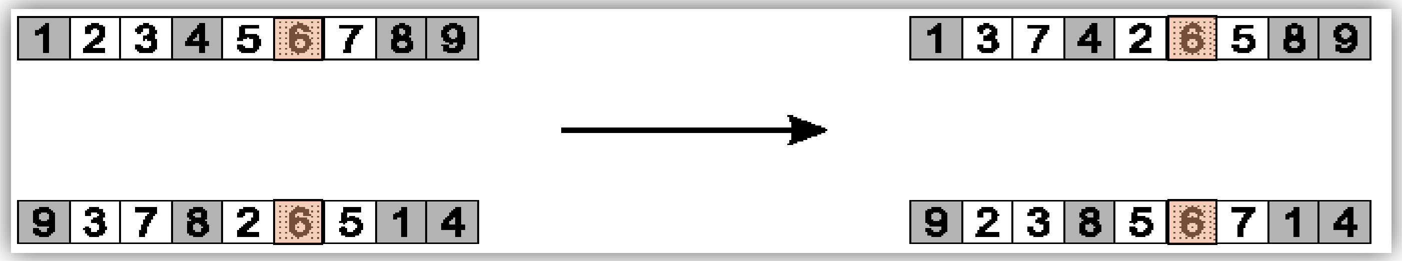
1. Make a cycle of alleles from P1 in the following way.
   (a) Start with the first allele of P1.
   (b) Look at the allele at the *same position* in P2.
   (c) Go to the position with the *same allele* in P1.
   (d) Add this allele to the cycle.
   (e) Repeat step b through d until you arrive at the 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



Genetic Programming: Tree Representation

- Trees are a universal form, e.g. consider
- Arithmetic formula:

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

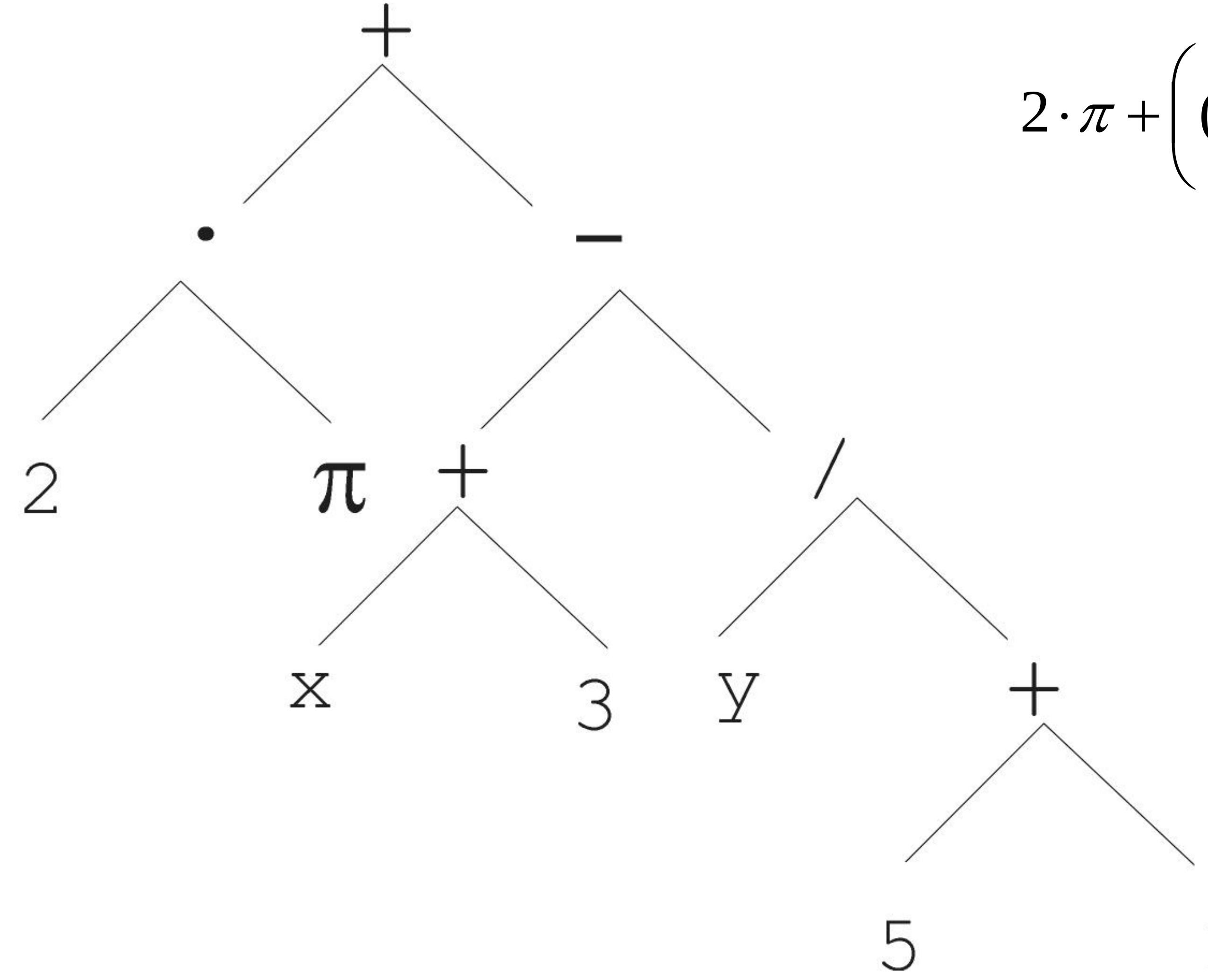
- Logical formula:

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

- Program:

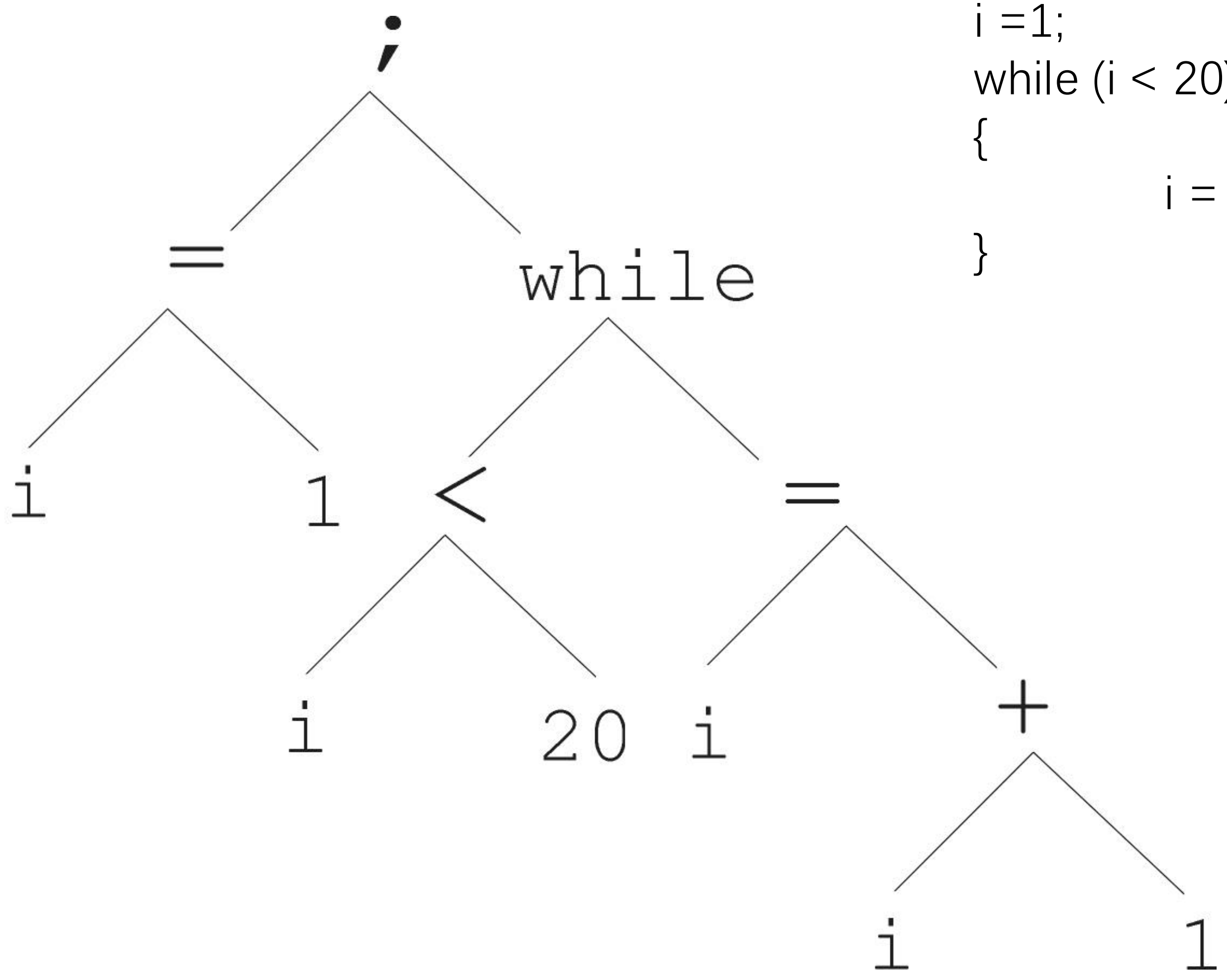
```
i =1;
while (i < 20)
{
        i = i +1
}
```

Genetic Programming: Tree Representation



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

Genetic Programming: Tree Representation

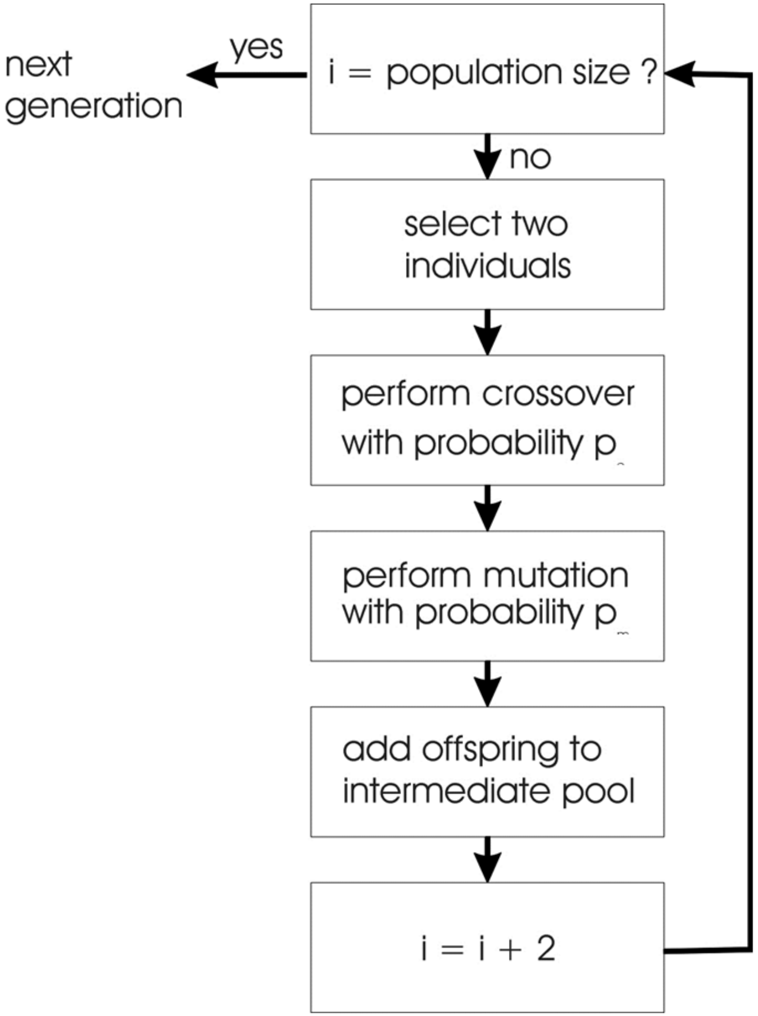



```
i =1;
while (i < 20)
{
        i = i +1
}
```

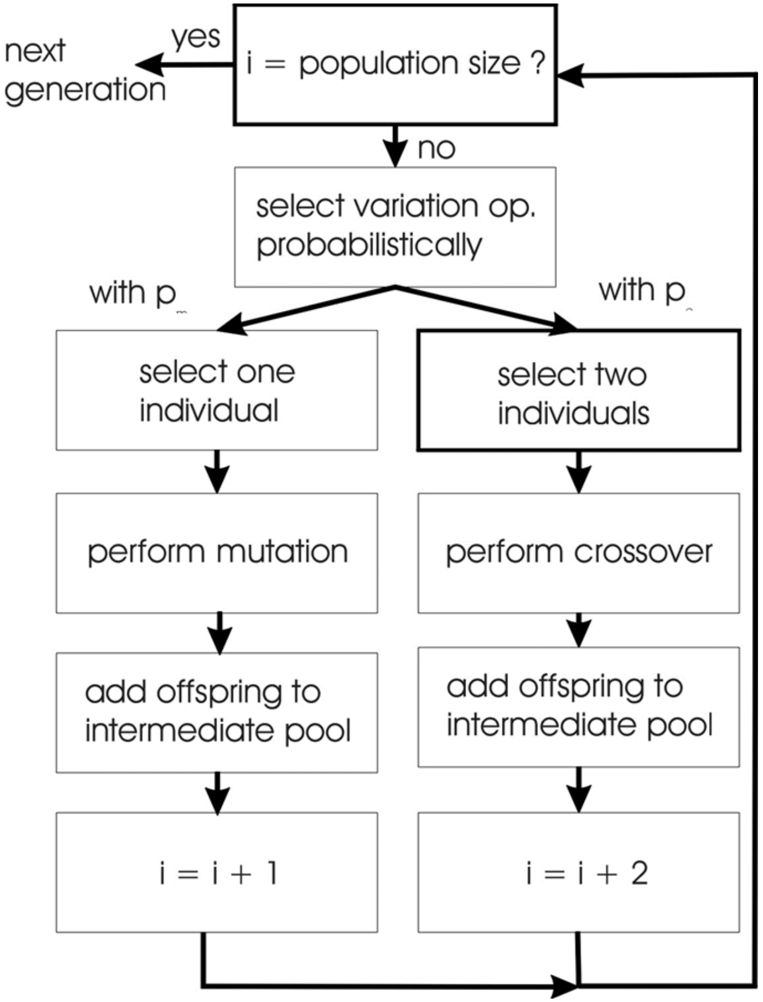
Genetic Programming: Tree Representation

- Tree shaped chromosomes are non-linear structures
- Trees in GP (Genetic Programming) may vary in depth and width

Genetic Programming: Variation Operators



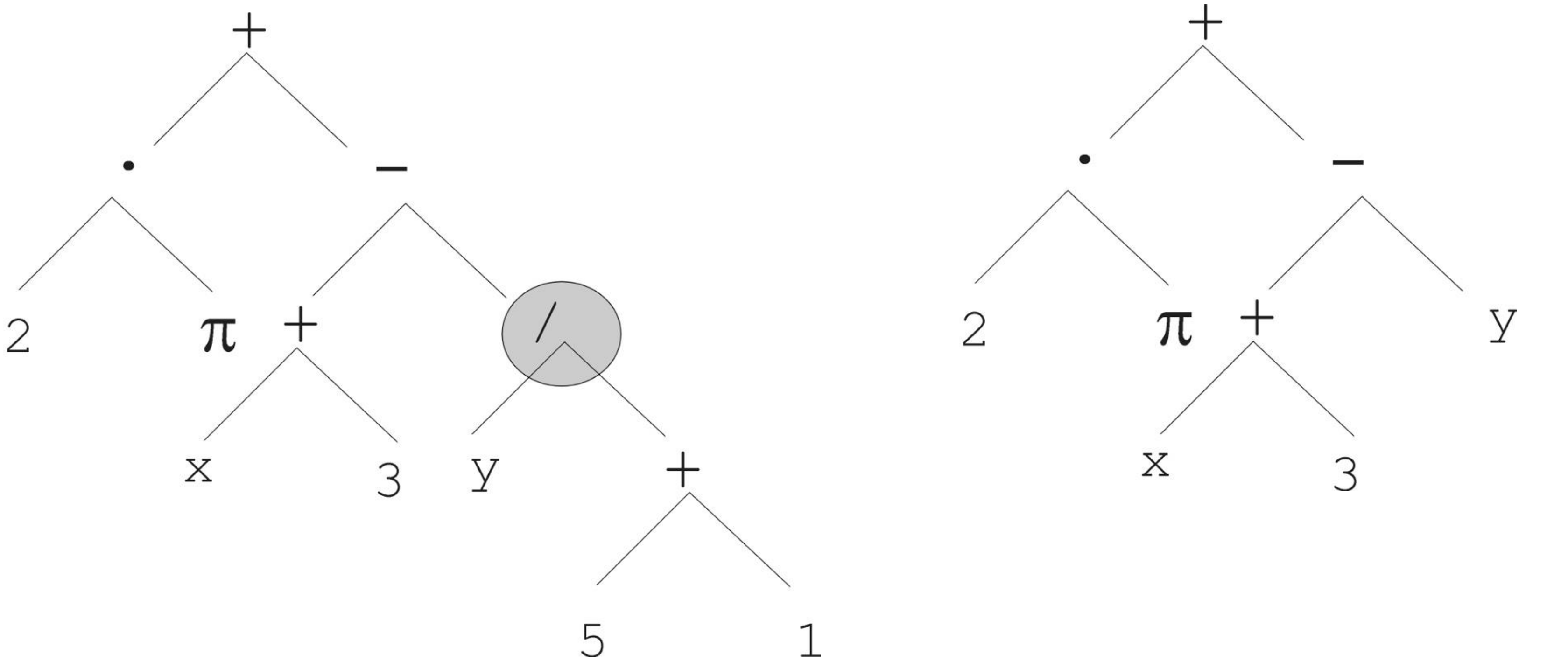
GA flowchart



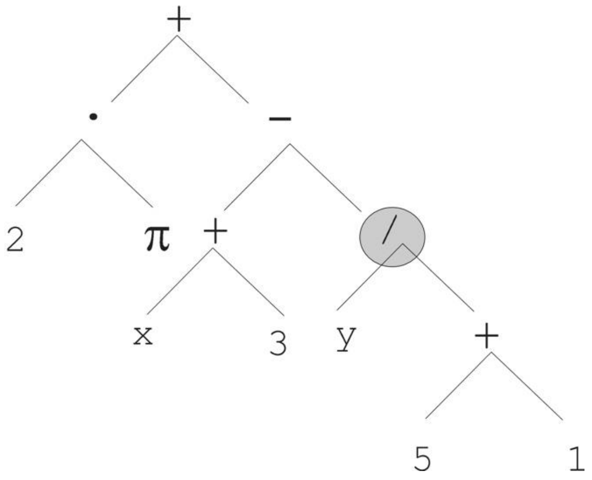
GP flowchart

Genetic Programming: Mutation

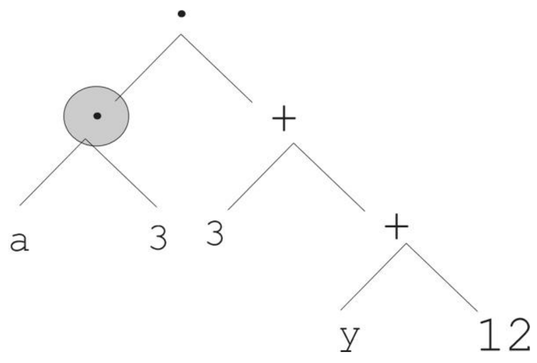
- Most common mutation: replace randomly chosen subtree by randomly generated tree



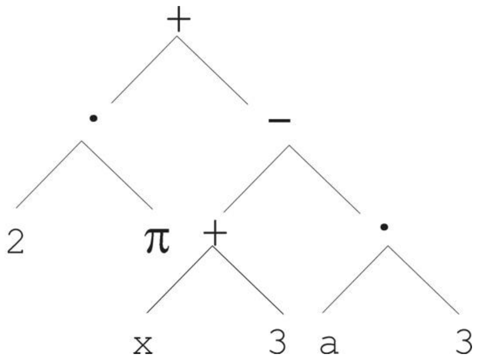
Genetic Programming: Recombination



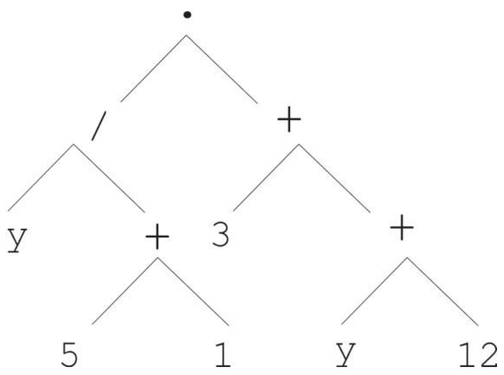
Parent 1



Parent 2

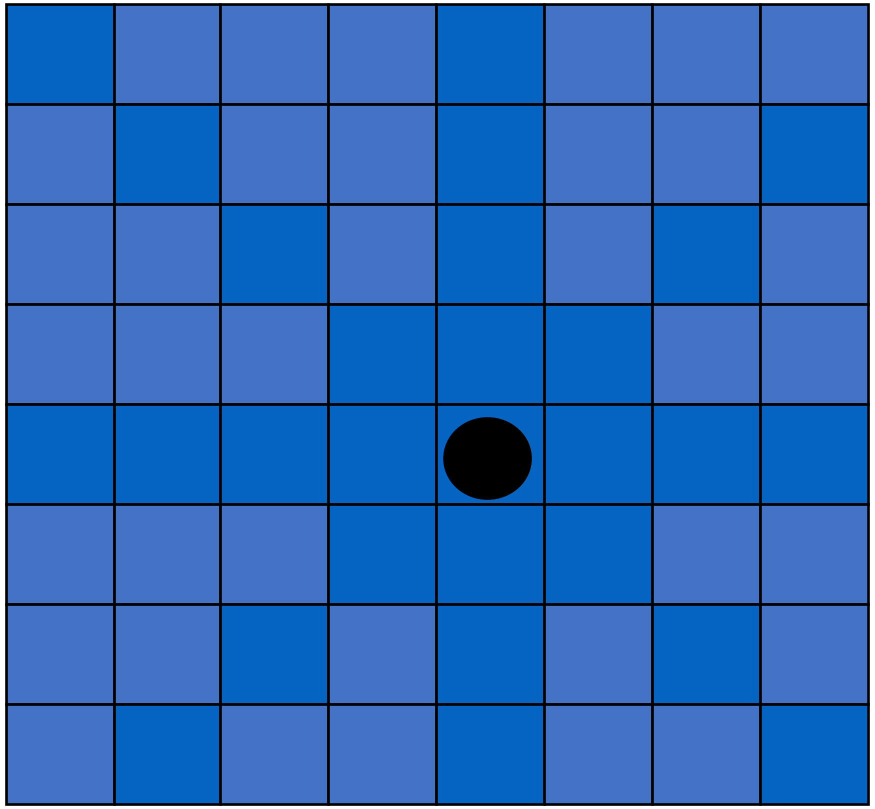


Child 1



Child 2

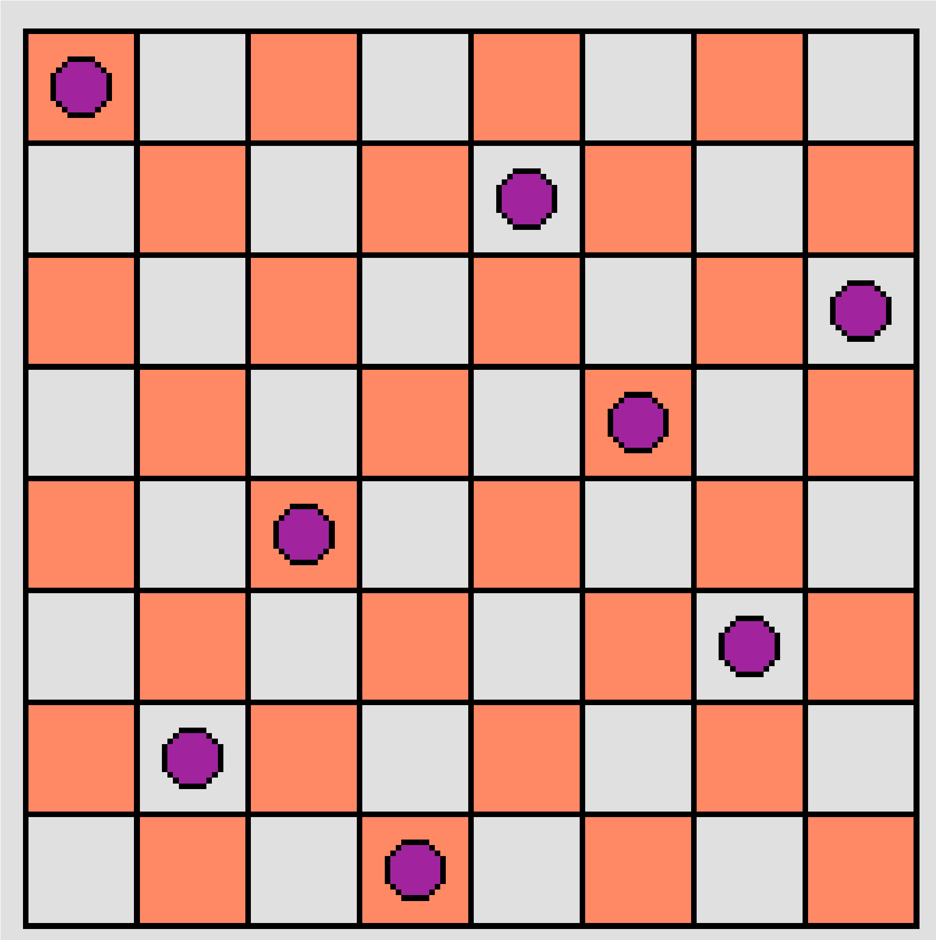
Example:
The 8-queens problem



- Representation?
- Fitness function?
- Variation operators?
- Selection operators?

Place 8 queens on an 8x8 chessboard in such a way that they cannot check each other

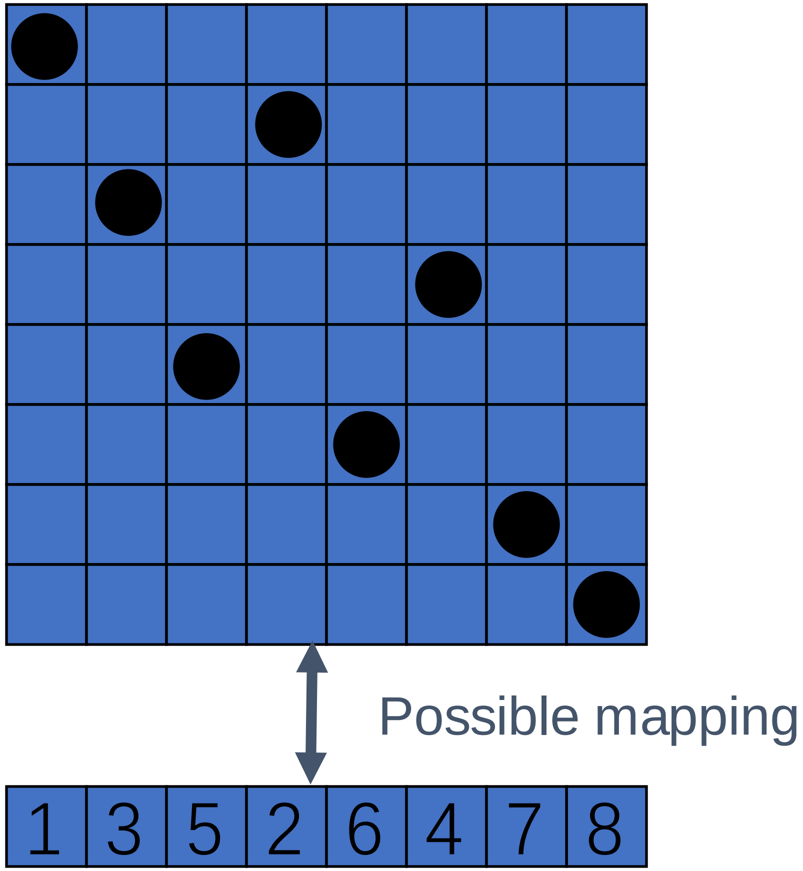
Example:
The 8-queens problem – one solution



The 8-queens problem: Representation

Phenotype:
a board configuration

Genotype:
a permutation of
the numbers 1–8



The 8-queens problem: Fitness evaluation

- Penalty of one queen: the number of queens she can check
- Penalty of a configuration: the sum of penalties of all queens
- Note: penalty is to be minimized
- Fitness of a configuration: inverse penalty to be maximized

The 8-queens problem: Mutation

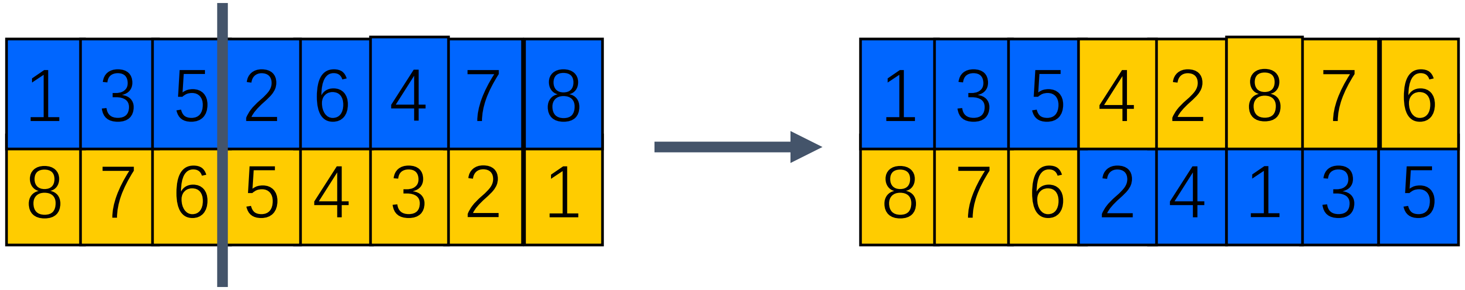
Small variation in one permutation, e.g.:

- swapping values of two randomly chosen positions,

The 8-queens problem: Recombination

Combining two permutations into two new permutations:

- choose random crossover point
- copy first parts into children
- create second part by inserting values from other parent:
  - in the order they appear there
  - beginning after crossover point
  - skipping values already in child



The 8-queens problem: Selection

- Parent selection:
  - Pick 5 random parents and take best 2 to undergo crossover
- Survivor selection (replacement)
  - Merge old (parents) and new (offspring) population
  - Throw out the 2 worst solutions

Concluding Insights: Evolutionary Algorithms

- EAs provide efficient, effective techniques for optimization and machine learning applications.
- They are generally effective in **discrete optimization** problems with a **continuous fitness landscape.**
- Simulate natural selection, where the population is composed of *candidate solutions*.
- Focus is on evolving a population from which strong and diverse candidates can emerge via mutation and crossover (mating).

Concluding Insights: Evolutionary Algorithms

- How unrealistic are Evolutionary Algorithms as representations of biological evolution?
- Are computer scientists truly inspired by evolutionary theory?