

---

# OPTIMIZATION FOR AI

## GLOBAL AND MULTI-OBJECTIVE OPTIMIZATION

---

Luca Manzoni

---

---

# TEAM OF THE COURSE

---

Code:  
**mhs13n**

The slides and material of the lectures will be  
uploaded on the Teams of the course

---



---

# EXAM STRUCTURE

---

- **Project + oral exam**
  - Project should be sent about a week before the oral exam
  - There will be some fixed dates for the exam (including one on December)
  - Exams outside the schedule are possible (if for a reason or if the next date is too far in the future)
-



---

# EXAM STRUCTURE

---

- Kinds of projects:
    - Implement an existing paper and reproduce the results
    - Apply evolutionary algorithms / swarm intelligence algorithms to an existing problem
    - Literature review on a specific topic
  - A set of projects will be presented later in the course, but we can discuss personalized project
-



---

# EXAM STRUCTURE

---

- Oral exam:
    - First part: presentation (15-20 minutes max) of your project
    - Second part: questions about all topics of the course
  - Both parts are essential!
-



---

# EXAM CHECKLIST

---

## ☒ **Project**

- ☒ Select your project
- ☒ Do your project
- ☒ Fix a date for the oral exam
- ☒ Send the project for evaluation one week before the oral exam

## ☒ **Oral Exam**

- ☒ Present your project
  - ☒ Answer questions on all the topics of the course
-



---

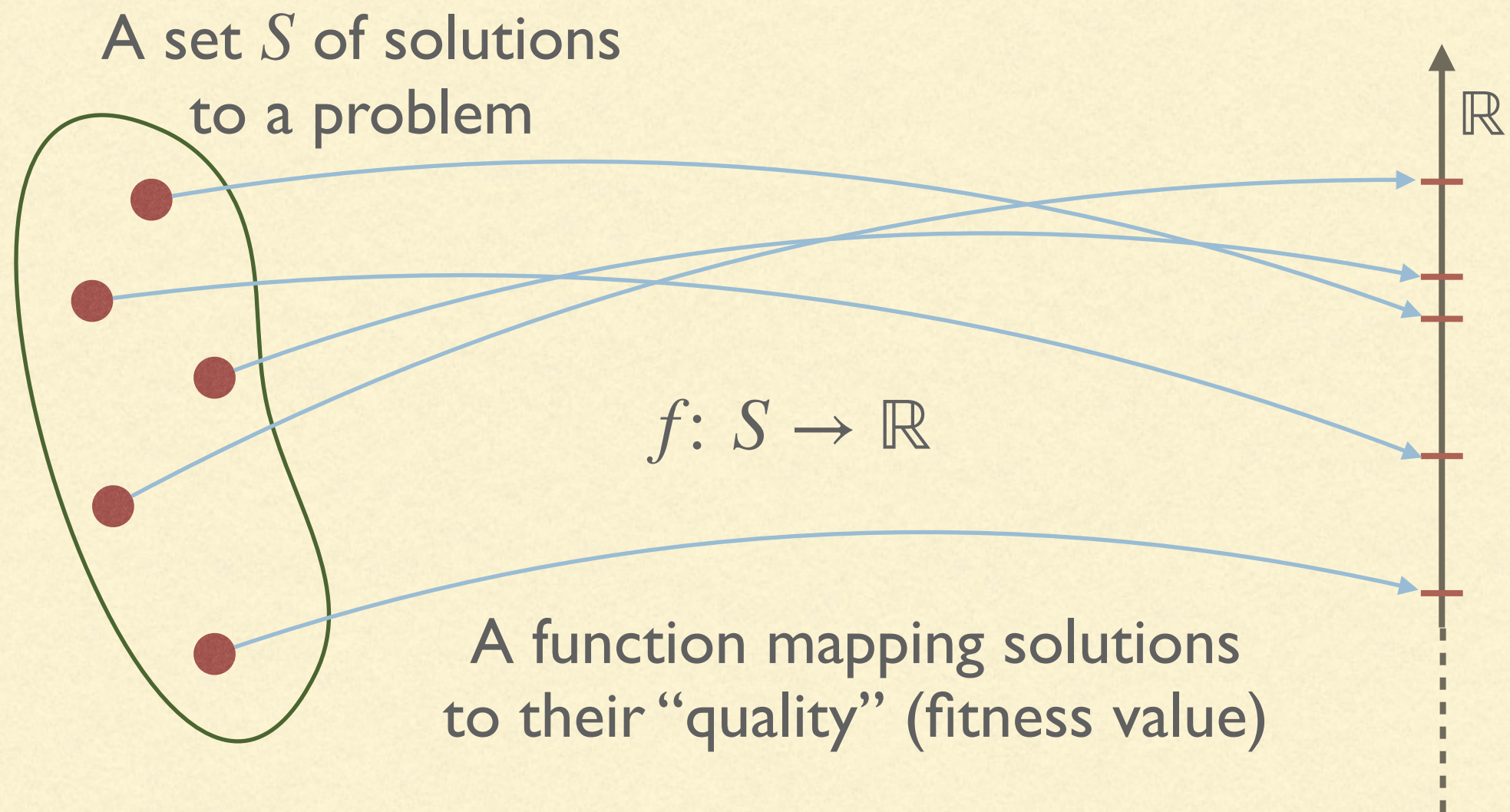
# OUTLINE

---

- What are population-based optimization methods
  - Outline of the course
  - Reference material
  - Genetic Algorithms
  - Evolution Strategies
-



# WHAT ARE WE TALKING ABOUT?



We want to find  $\operatorname{argmax}_{x \in S} f(x)$  or  $\operatorname{argmin}_{x \in S} f(x)$



---

# HOW CAN WE FIND THE OPTIMUM?

---

- We might be unable to solve the problem analytically
  - $S$  might be too large to perform an exhaustive search
  - We might have very few assumption on  $f$ , i.e.,  $f$  is a “black box”
  - It might be OK to return “good enough” solutions
-



---

# A TRIVIAL PROBLEM: ONEMAX

---

- Let  $S = \{0,1\}^n$ , hence the size of the space is  $2^n$
  - Let  $f(x) = \#$  of ones in  $x$
  - We want to maximise  $f$
  - Clearly, the optimum is  $1^n$  with fitness value  $n$
-



---

# APPROACH #1: RANDOM SEARCH

---

- Select a solution  $b$  from  $S$  (it is not important how)
  - Repeat the following until some termination criteria is met
    - Let  $x$  be a solution selected uniformly at random from  $S$
    - If  $f(x) \geq f(b)$  substitute  $b$  with  $x$
  - Return  $b$
-



---

# APPROACH #1: RANDOM SEARCH

---

- Even if we avoid sampling the same solution more than once, we will still need to explore a significant fraction of the search space
  - Without repetitions, it is like exhaustive search for some choice of the order of enumeration of the solutions
  - Generally unfeasible
  - In **some** search spaces this is better than other kinds of search...
  - ...but hopefully this is not something that happens for real problems
-



---

# APPROACH #2: HILL CLIMBING

---

- Let  $b$  be an initial solution from  $S$
  - Repeat the following until some termination criteria is met
    - Let  $x$  be a **neighbor** of  $b$
    - If  $f(x) \geq f(b)$  then replace  $b$  with  $x$
  - Return  $b$
-



---

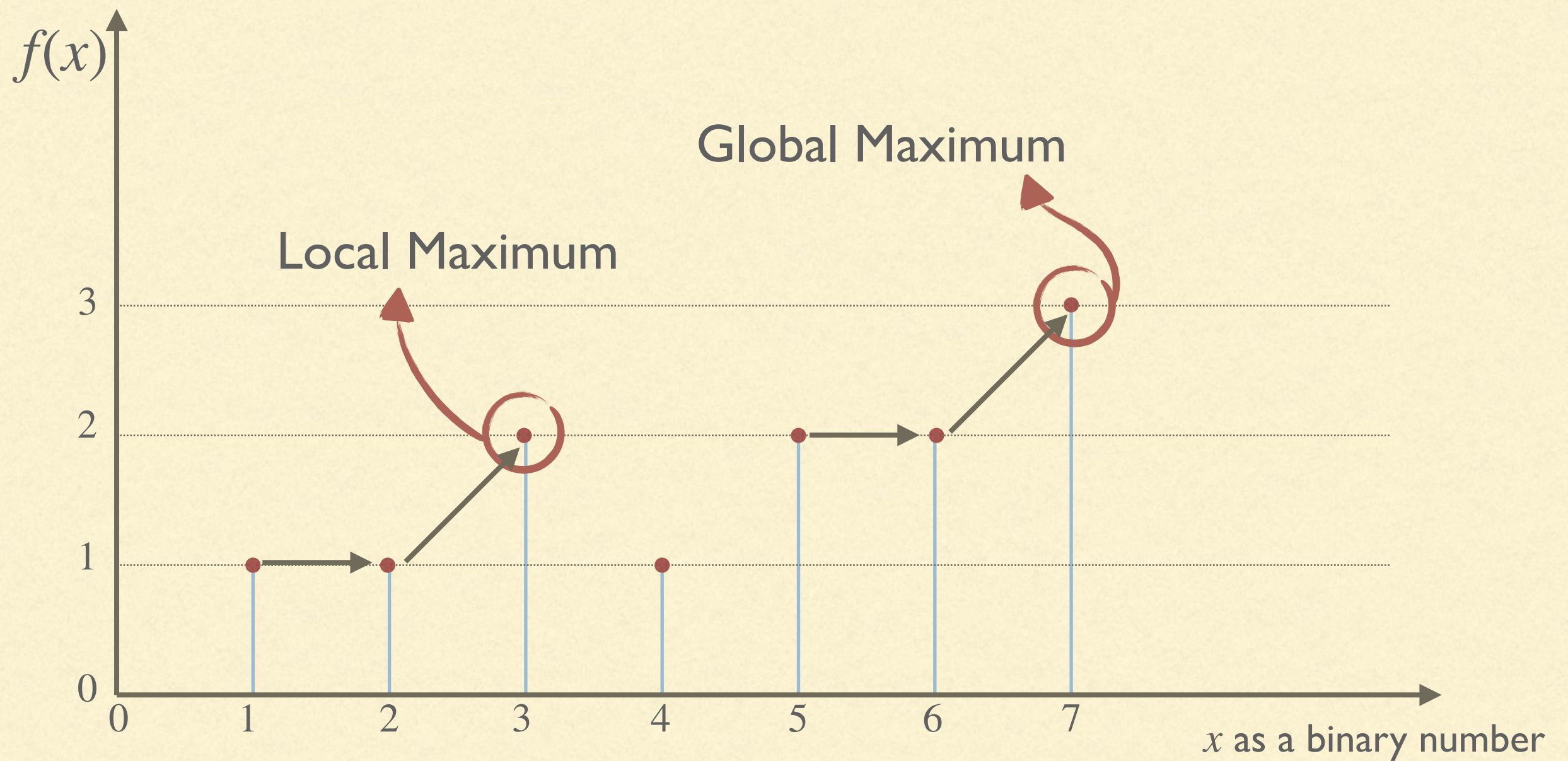
# APPROACH #2: HILL CLIMBING

---

- We are defining a neighborhood structure on  $S$
  - The ability to find a global optimum depends on this structure:
    - OneMax with neighborhood of  $x$  given by  $x + 1$  and  $x - 1$
    - OneMax with neighborhood of  $x$  given by all binary strings at Hamming distance one.
-



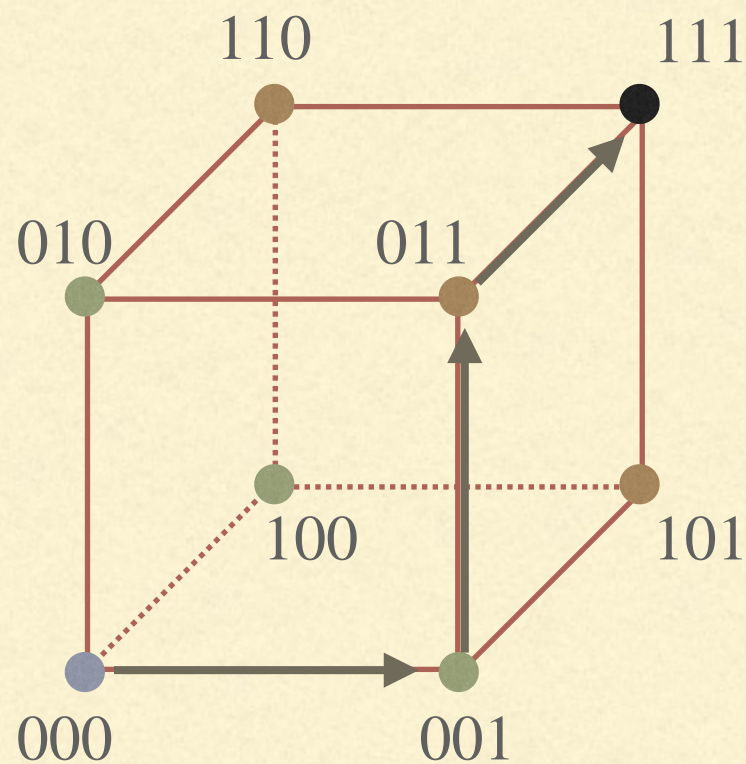
# LOCAL AND GLOBAL MAXIMA





# LOCAL AND GLOBAL MAXIMA

$f(x)$	
●	3
●	2
●	1
●	0



Notice that there are  
no local optima



---

# APPROACH #3: SIMULATED ANNEALING

---

- Let  $b$  be an initial solution from  $S$  and  $T$  the “temperature”
  - Repeat the following until some termination criteria is met
    - Let  $x$  be a **neighbor** of  $b$
    - If  $f(x) \geq f(b)$  then replace  $b$  with  $x$
    - Otherwise replace  $b$  with  $x$  with probability  $e^{\frac{f(x) - f(b)}{T}}$
    - Update  $T$  (by decreasing it) according to some schedule
  - Return  $b$
-



---

# APPROACH #3: SIMULATED ANNEALING

---

- The idea is to allow the selection of less fit solutions with some probability that depends from the time and the difference in fitness
  - This allows to reduce the risk of getting stuck in a local optimum
  - The choice of the schedule is important!
-



---

# MULTIPLE RESTARTS

---

- For Hill Climbing and Simulated Annealing we can reduce the risk of getting stuck on a local optimum by repeating the process several times
  - Each repetition is independent...
  - ...can we do better?
  - The idea is to use a multiset of solutions instead of working with one solution at a time
  - This time the different solutions “interact” in some way
-



---

# OUTLINE OF THE COURSE

---

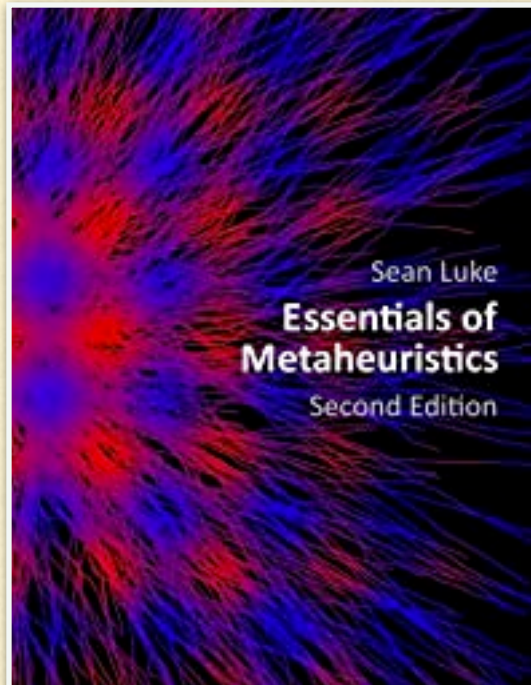
- Genetic Algorithms
  - Evolution Strategies
  - Genetic Programming
  - Particle Swarm Optimization and Ant-Colony Optimization
  - Differential Evolution
  - Neuroevolution
  - EDA and CMA-ES
  - Parallel implementations
  - Multi-objective optimization
  - Coevolution
  - Policy Optimization
  - Theory of Evolutionary Computation
-



---

# REFERENCE MATERIAL

---

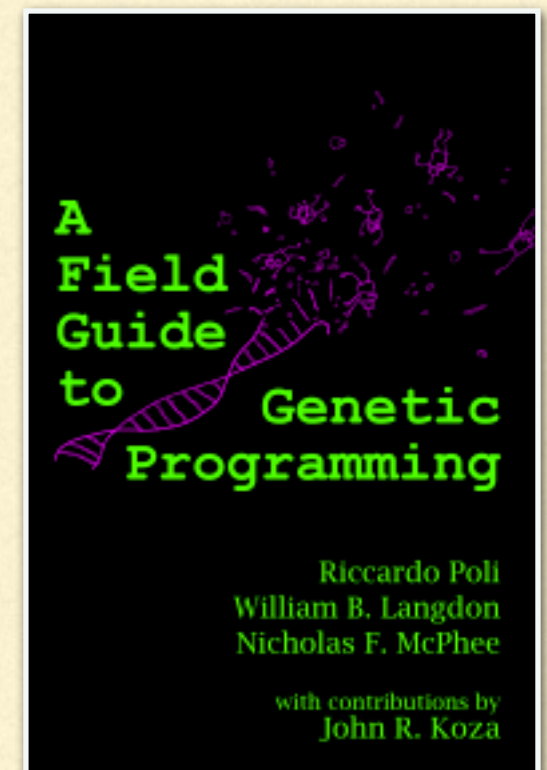


S. Luke

*Essentials of Metaheuristics, 2nd Edition*

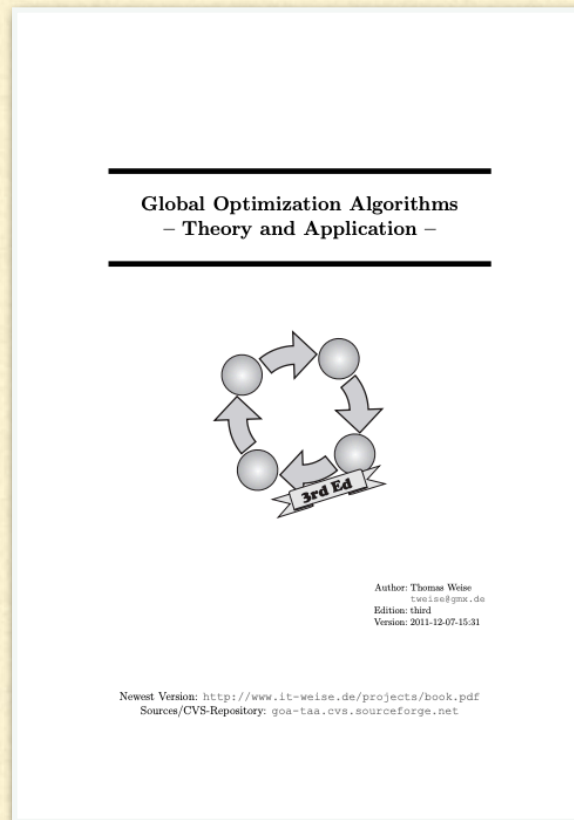
<https://cs.gmu.edu/~sean/book/metaheuristics/>

R. Poli, W. R. Langdon, N. F. McPhee  
*A Field Guide to Genetic Programming*  
<http://www.gp-field-guide.org.uk>



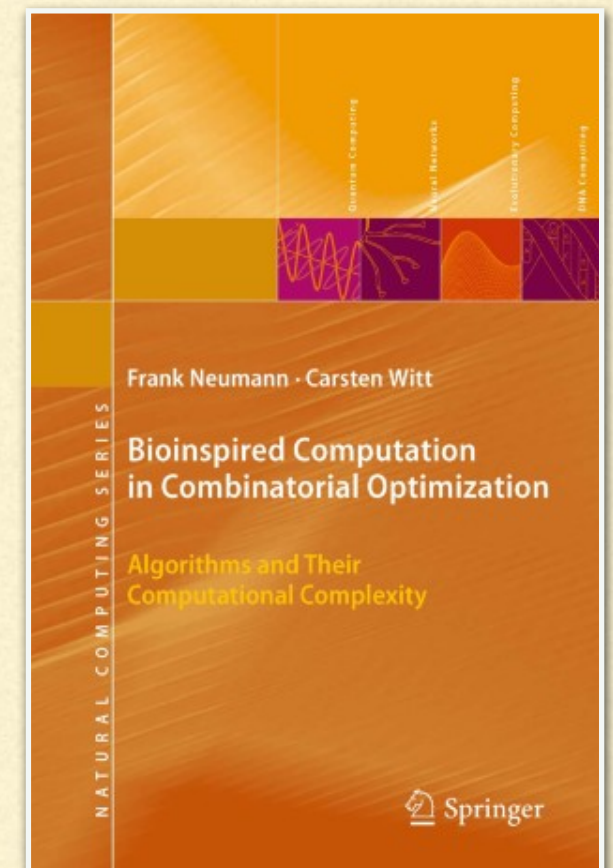


# REFERENCE MATERIAL



T. Weise

*Global Optimization Algorithms – Theory and Application, 3rd Edition*  
(Search for it online)



F. Neumann and C. Witt

*Bioinspired Computation in Combinatorial Optimization  
Algorithms and Their Computational Complexity*

<http://www.bioinspiredcomputation.com/self-archived-bookNeumannWitt.pdf>



---

# GENETIC ALGORITHMS

---



---

# GENETIC ALGORITHMS

---

- Invented by John Holland in the Seventies:  
John Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975
  - Inspired by the Darwinian theory of evolution
  - A multiset of solutions (called *population*, each solution is an *individual*) is iteratively evolved, with each iteration consisting of
    - Selection
    - Crossover and mutation
-



---

# GENETIC ALGORITHMS

---

- Solutions are usually represented as binary strings of fixed length
  - This is not a requirement, as we will see in one of the next lectures
  - We must now distinguish between the **genotype** and the **phenotype** of a solution, since some operators work on the genotype and some on the phenotype
-



---

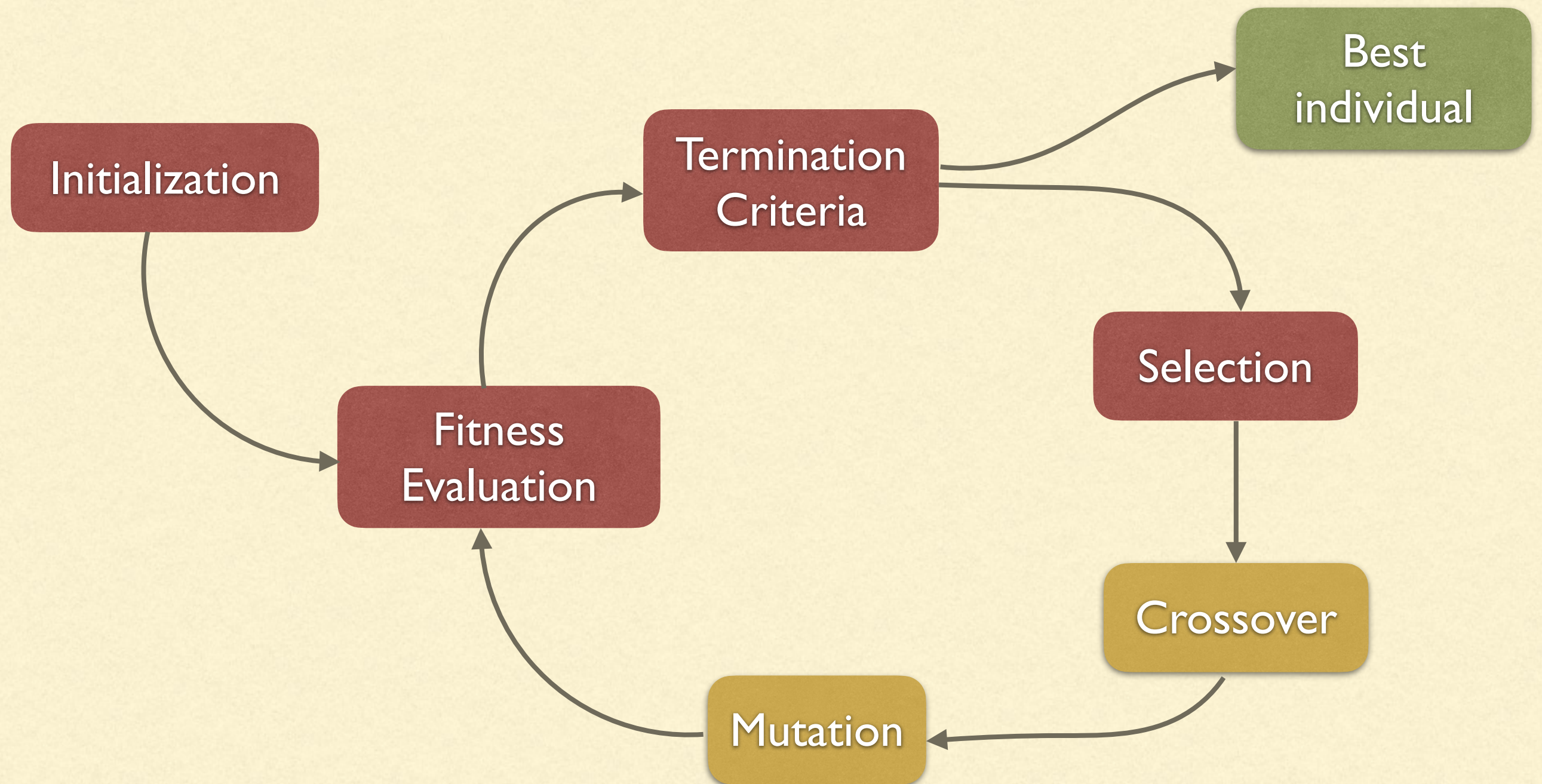
# PHENOTYPE AND GENOTYPE

---

- The **genotype** is how a solution is represented.  
E.g., a string of bits.
    - Crossover and mutation works on the genotype of a solution
  - The **phenotype** is the actual solution. E.g., the number of ones, the number represented by a string of bits, etc.
    - The fitness computes how good the phenotype of a solution is, without considering the genotype
    - Selection operates only on the phenotype ignoring the genotype
-



# EVOLUTION CYCLE

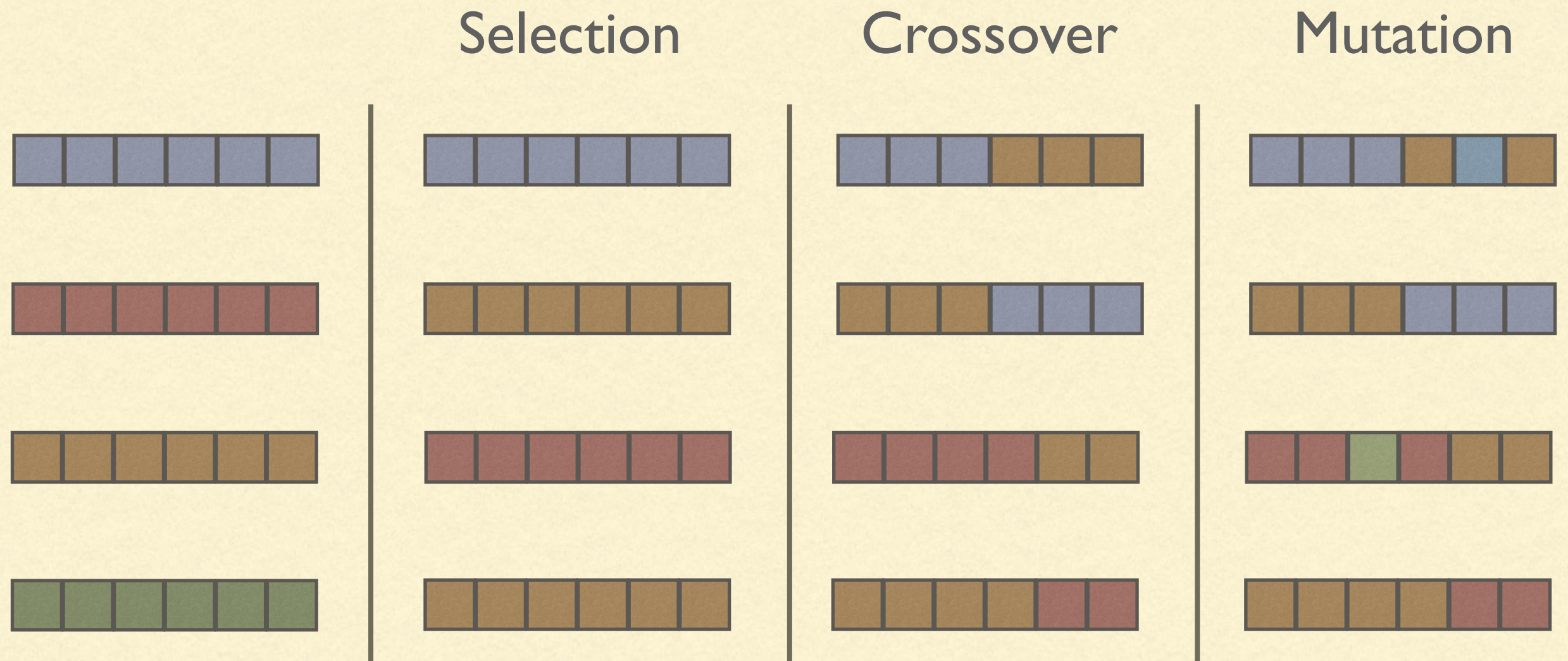




---

# A GA GENERATION

---





---

# GA PARAMETERS

---

- Population size  $N$
  - Type of selection
  - Type of crossover
  - Crossover probability  $p_{\text{cross}}$ , usually near one
  - Type of mutation
  - Mutation probability  $p_{\text{mut}}$ , usually small
  - Elitism or others additional modifications
-



---

# SELECTION METHODS

---

- There are multiple ways to decide which individuals are selected:
    - Roulette wheel selection
    - Ranked selection
    - Tournament selection
-




---

# ROULETTE WHEEL SELECTION

---

The probability of an individual to be selected is proportional to its fitness

$$p_{x,P} = \frac{f(x)}{\sum_{y \in P} f(y)}$$



Probability of  $x$  getting selected given that the current population is  $P$ .

---



---

# ROULETTE WHEEL SELECTION

---

- **Pro:** easy to implement
  - **Cons:** if an individual has a very large fitness w.r.t. the others in the population then we may reduce the diversity in the population by continuously selecting it
  - We may want something that depends on the *ranking* of the fitnesses, not on their raw value
-



---

# RANKED SELECTION

---

- We compute the fitness of all individuals in the population
  - The individuals are ranked w.r.t. their fitness values
  - Each rank has a fixed probability of being selected
    - E.g., the individual with the  $i^{\text{th}}$  best fitness will be selected with probability  $\frac{N - i + 1}{r_{\text{sum}}}$  where  $r_{\text{sum}}$  is a normalisation factor
-



---

# TOURNAMENT SELECTION

---

- Tournament selection is the most used kind of selection in GA
  - Let  $t \in \mathbb{N}^+$  be the *tournament size*
  - Extract  $t$  individuals from the current population (with reinsertion)
  - Select the one with the best fitness
-



---

# TOURNAMENT SELECTION

---

- Has the advantage of providing a “ranked” selection without specifying the function to assign the probabilities
  - The “selection pressure” (how difficult it is for a non-fit individual to be selected) can easily be tuned via the tournament size
    - Large tournament size → high selection pressure
    - Low tournament size → low selection pressure
-



---

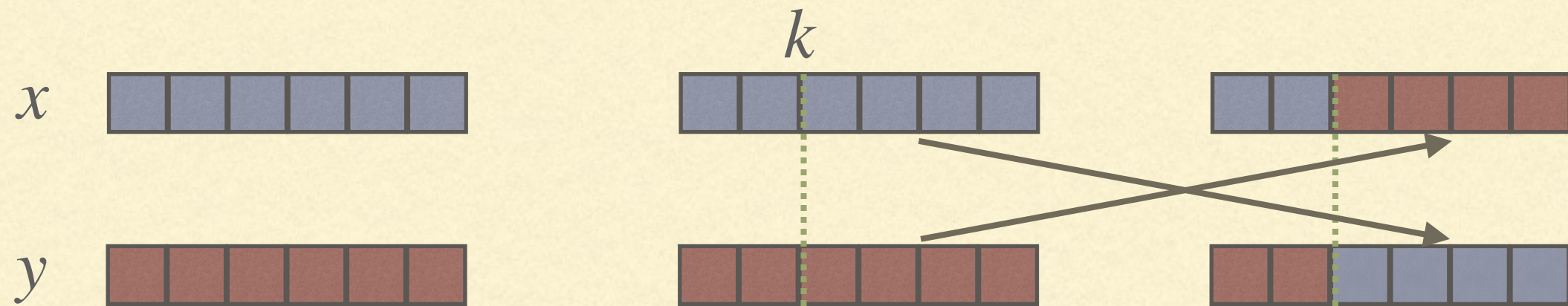
# CROSSTOVERS

---

- Some common crossovers:
    - One-point crossover
    - $m$ -points crossover
    - Uniform crossover
-



# ONE-POINT Crossover



- Given the two parent  $x = x_1x_2\cdots x_n$  and  $y = y_1y_2\cdots y_n$
- A crossover point  $k \in \{1, \dots, n\}$  is selected uniformly at random
- The two offsprings are  $x_1\cdots x_ky_{k+1}\cdots y_n$  and  $y_1\cdots y_kx_{k+1}\cdots x_n$



---

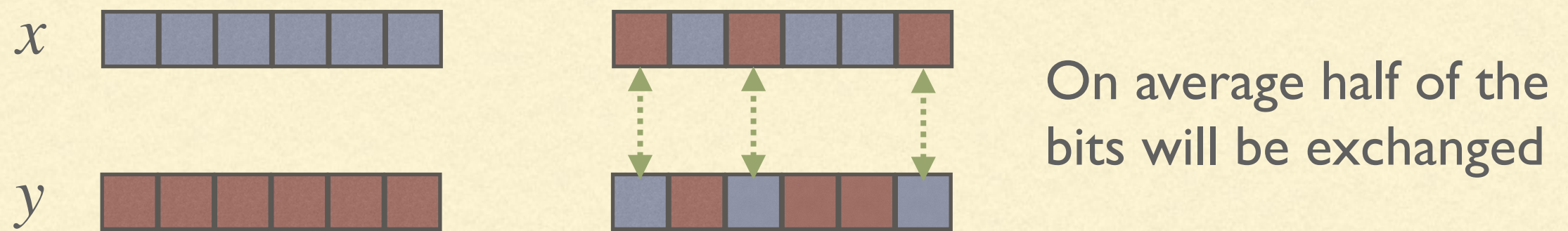
# *m*-POINTS CROSSOVER

---

- A direct generalization of one-point crossover where more than one point is selected
  - $k_1 \dots k_m \in \{1, \dots, n\}$  crossover points are selected (with  $k_i \leq k_{i+1}$  for all  $i \in \{1, \dots, m - 1\}$ )
  - The elements of the two parents are swapped between each pair of consecutive crossover points
-



# UNIFORM CROSSOVER



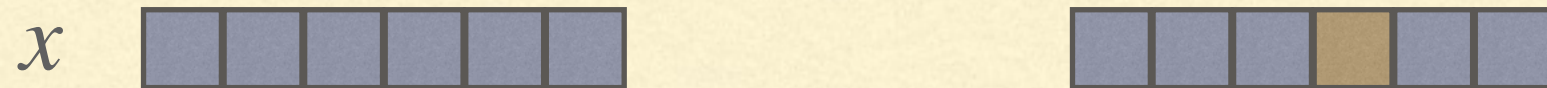
- Given the two parent  $x = x_1x_2\cdots x_n$  and  $y = y_1y_2\cdots y_n$
- For each  $i \in \{1, \dots, n\}$  with probability  $1/2$  the  $i^{\text{th}}$  element in the first (resp., second) offspring will be  $x_i$  (resp.,  $y_i$ ) and otherwise  $y_i$  (resp.,  $x_i$ )



---

# BIT-FLIP MUTATION

---



- Given an individual  $x = x_1x_2\cdots x_n$
  - For each  $i \in \{1, \dots, n\}$  flip the bit  $x_i$  with probability  $p_{\text{mut}}$
  - Usually  $p_{\text{mut}} = \frac{1}{n}$  to mutate (on average) one bit per individual
-



---

# CROSSOVER AND MUTATION

---

- Crossover is **not** “global” mutation
  - That is, given two Boolean vectors  $x$  and  $y$  it is not possible to obtain every possible vector from them. Why?
  - If  $x_i = y_i = 1$  then the result of crossover will never have  $x_i = 0$
  - This will be important for the definition of the topological properties of crossover and mutation
-



---

# EXPLOITATIVE VARIANTS

---

- **Elitism.** A certain fraction of the individuals with the best fitness are preserved in the next generation unless better solutions are found
    - Keep only the best individual found so far
    - Keep the top  $k$  individuals
    - Keep the top  $p\%$  of individual
  - Elitism changes the selection pressure: new individuals have better “competitors” in the next selection phase
-



---

# STEADY STATE GA

---

- Instead of creating a completely new population at each iteration we only replace some of the individuals in the existing population
  - Which individuals to replace?
    - The worst ones
    - A random selection
-



---

# HYBRID GA

---

- After each generation all individuals are improved with a local search algorithm (e.g., hill climbing)
  - Called “Memetic Algorithms”, better names are “Lamarckian algorithms” or “Baldwin Effect Algorithms”
  - Parameters:
    - How frequently the local search is performed
    - How many iterations of local search are performed each time
-



---

# REPRESENTATION

---

- **Beyond Binary.** It is possible to generalize GA to use symbols from a finite set/alphabet  $\Sigma$  instead of using only  $\{0,1\}$
  - The only difference is that mutation will have to select (usually uniformly at random) between  $|\Sigma| - 1$  symbols instead of simply flipping a bit
  - If there is an ordering between the elements of  $\Sigma$  (e.g.,  $\Sigma = \{0,1,2,3\}$ ) then mutation can be changed to be, for example “add or subtract one”.
-



---

# SPECIAL REPRESENTATIONS FOR GENETIC ALGORITHMS

---



---

# REAL-VALUED GA

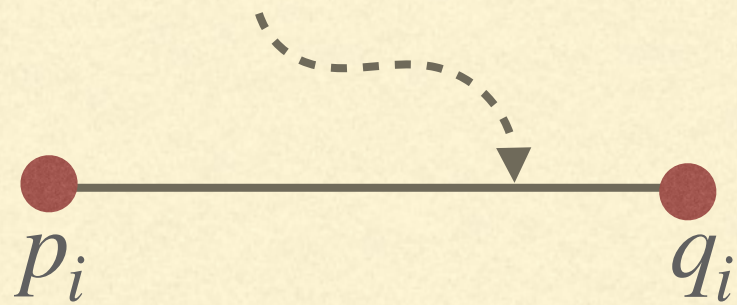
---

- Until now we have seen binary valued (or integer valued) GA
  - We can represent each floating point numbers as 32/64 binary genes...
  - ...but this means that different bits have different impact on the encoded number
  - If each gene is a floating point value then mutation and crossover should be adapted
-



# CROSSOVER

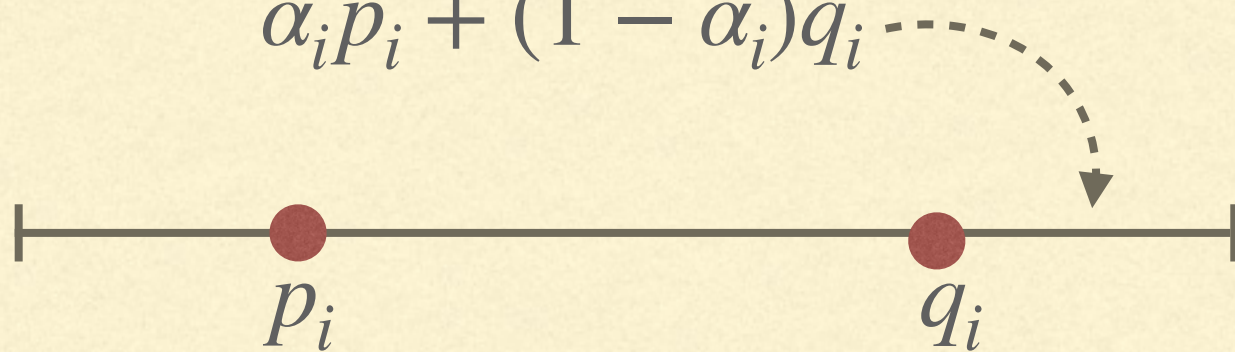
$$\alpha_i p_i + (1 - \alpha_i) q_i$$



Intermediate recombination

$$\alpha_i \leftarrow \text{random}(0,1)$$

$$\alpha_i p_i + (1 - \alpha_i) q_i$$



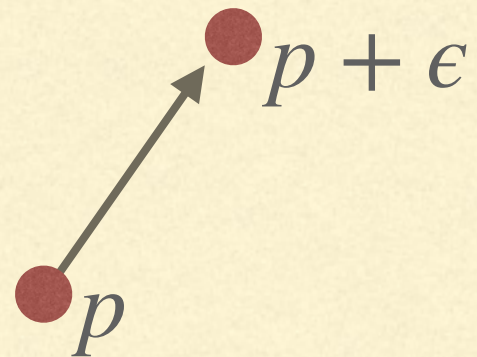
Line recombination

$$\alpha_i \leftarrow \text{random}(-k, 1 + k)$$



# MUTATION

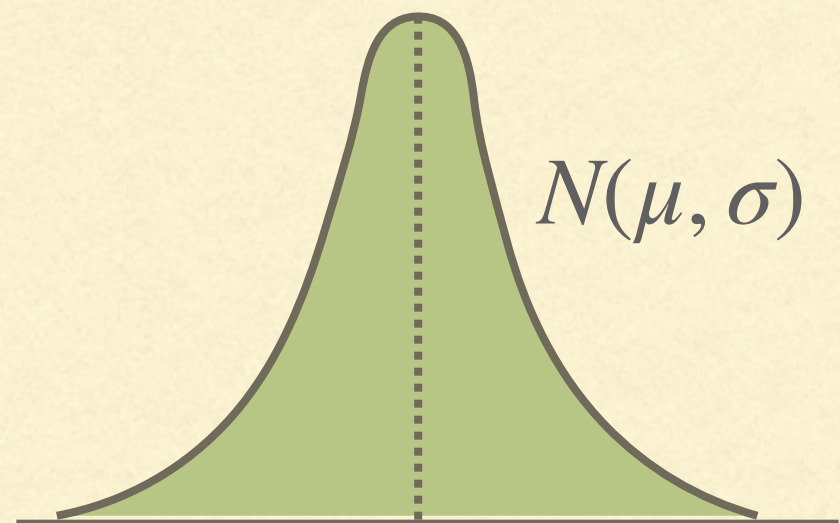
Add a small value  
to each coordinate  
of the individual



Uniform between two values



Gaussian





---

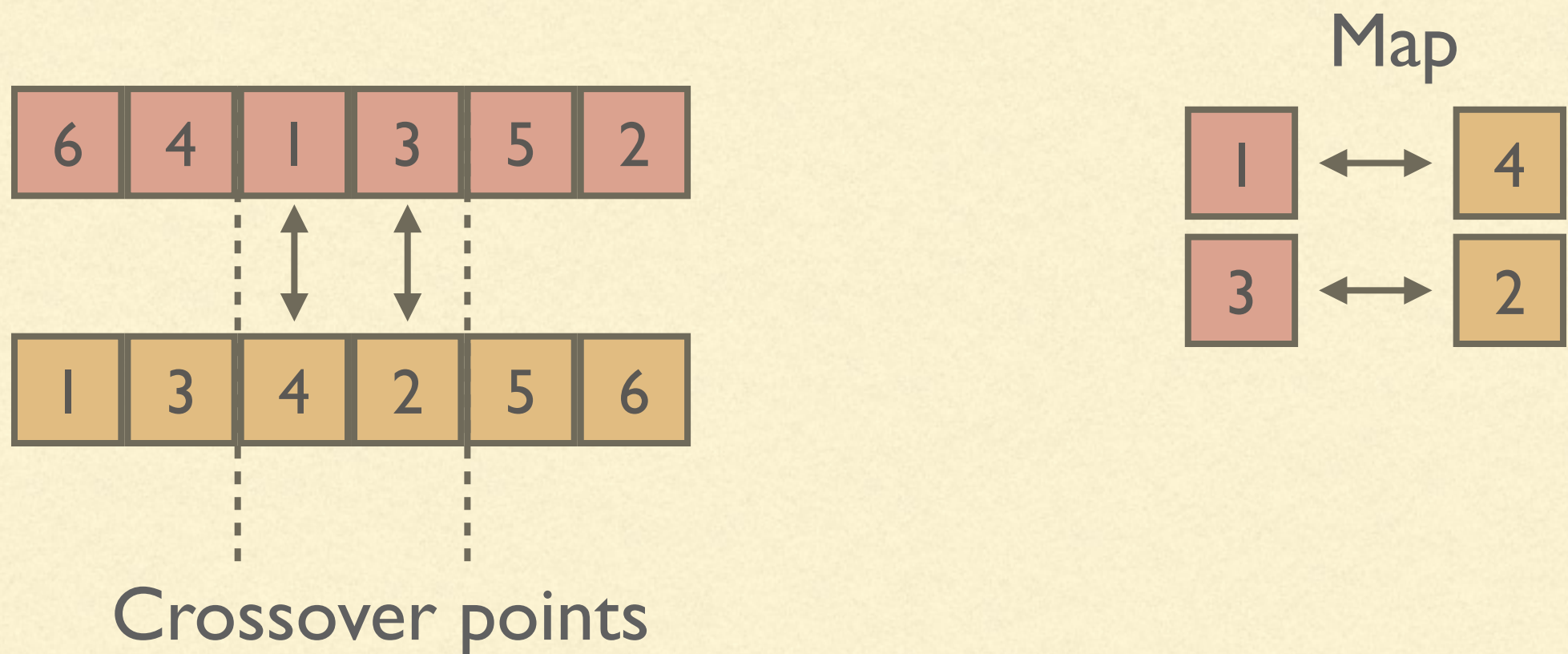
# CYCLE AND PMX CROSSTOVERS

---

- Sometimes we need additional constraints in the representation of an individual by GA
  - One usual constraint is that each individual must be a permutation of the numbers from 1 to  $n$
  - Mutation can be performed by swapping two positions
  - Traditional crossover usually do not respect the constraints
-



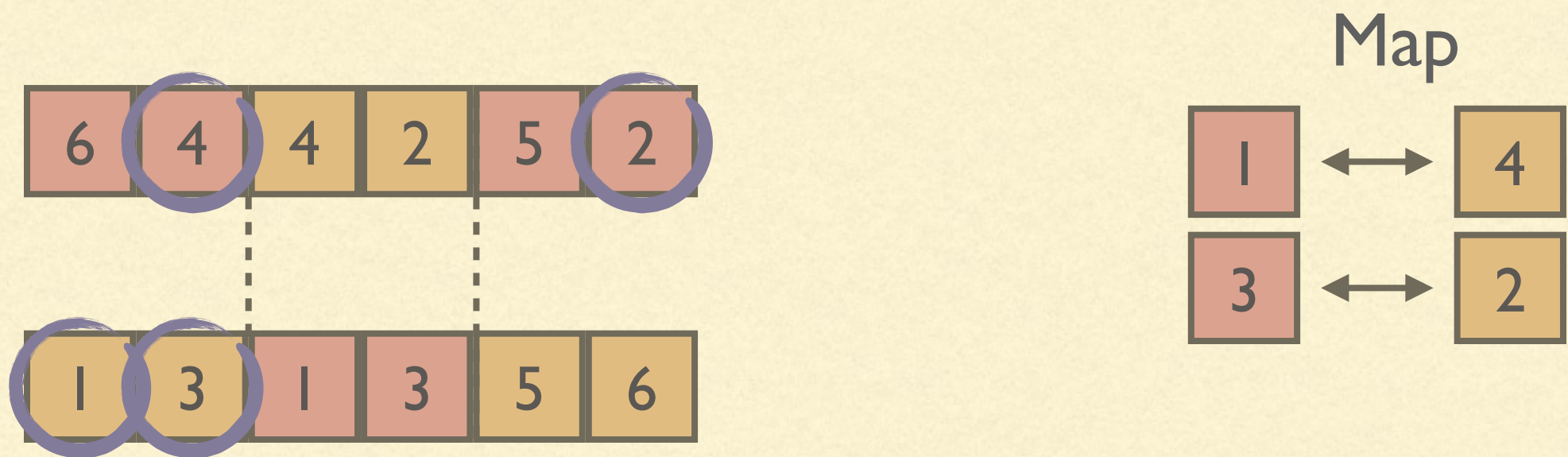
# PARTIALLY MAPPED CROSSOVER (PMX)



We select two crossover point and we build a map



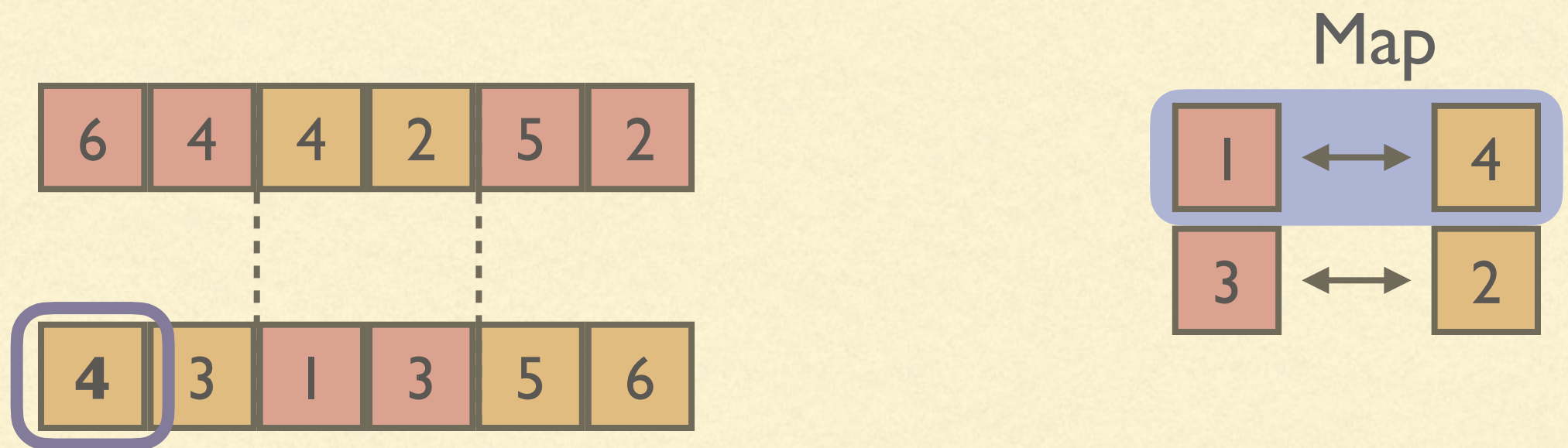
# PARTIALLY MAPPED CROSSOVER (PMX)



We perform the exchange but the offspring are not valid



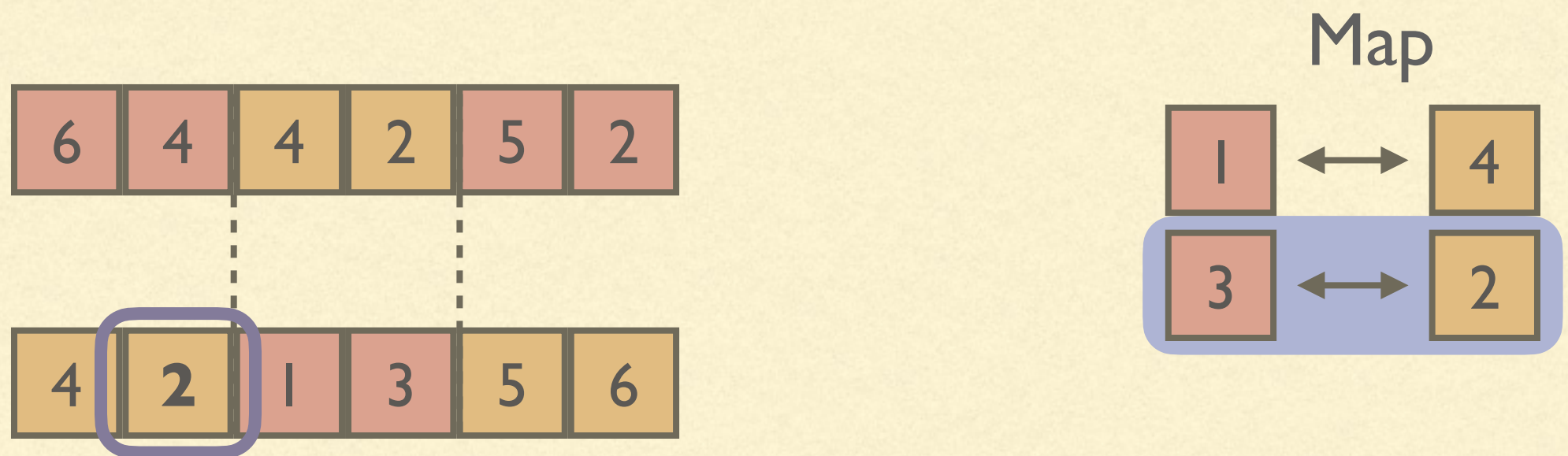
# PARTIALLY MAPPED CROSSOVER (PMX)



We iterate the map until we have resolved the conflicts



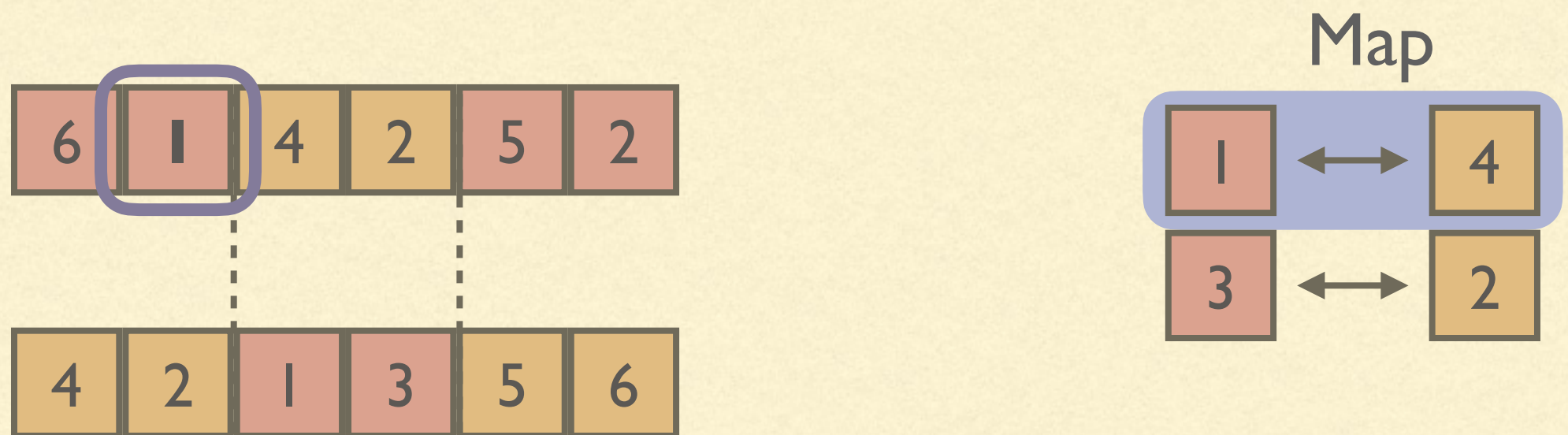
# PARTIALLY MAPPED CROSSOVER (PMX)



We iterate the map until we have resolved the conflicts



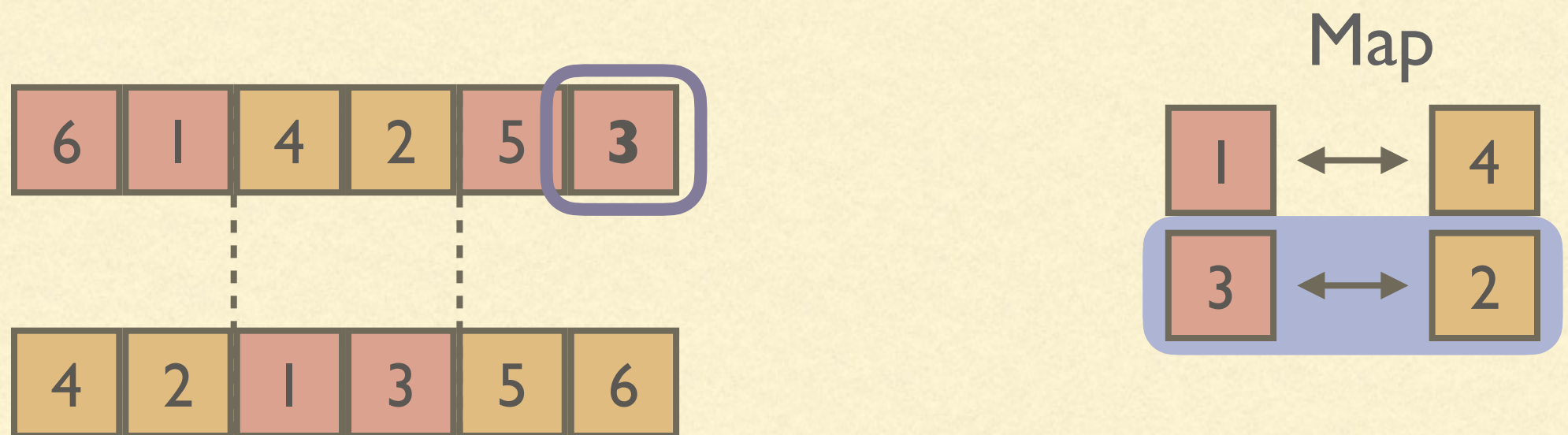
# PARTIALLY MAPPED CROSSOVER (PMX)



We iterate the map until we have resolved the conflicts



# PARTIALLY MAPPED CROSSOVER (PMX)



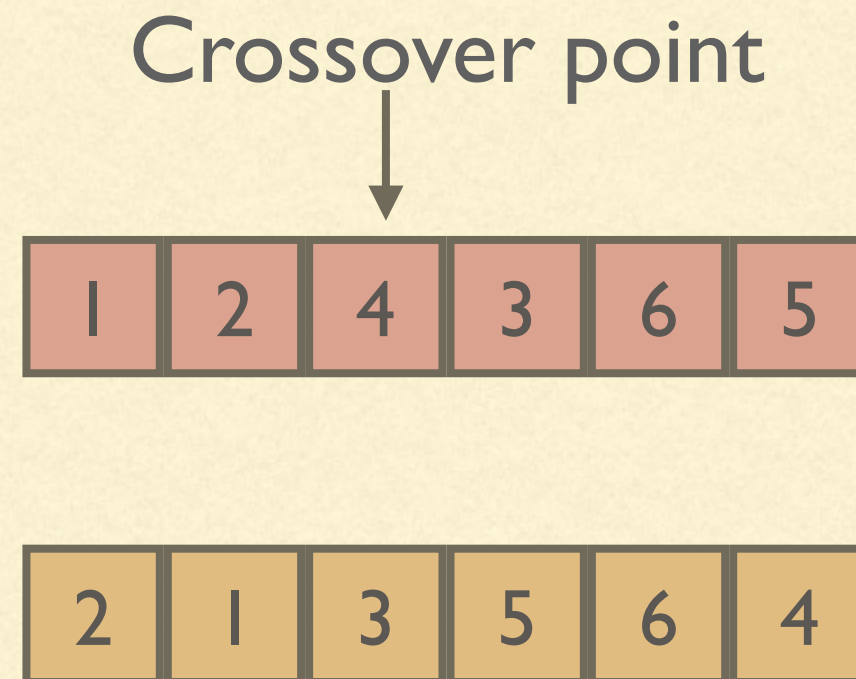
We iterate the map until we have resolved the conflicts



---

# CYCLE CROSSOVER

---



We select a single starting point  
and we search the same value in the second parent

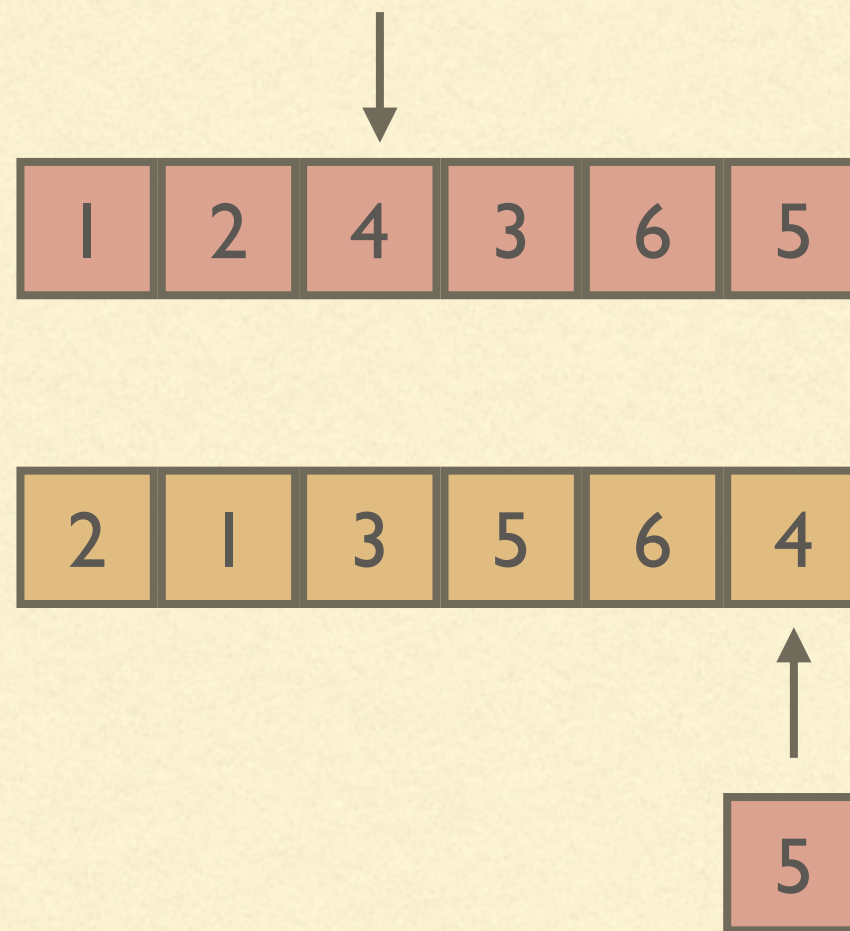
---



---

# CYCLE CROSSOVER

---



Once found we copy the value from the first parent.  
Repeat until we return to the beginning

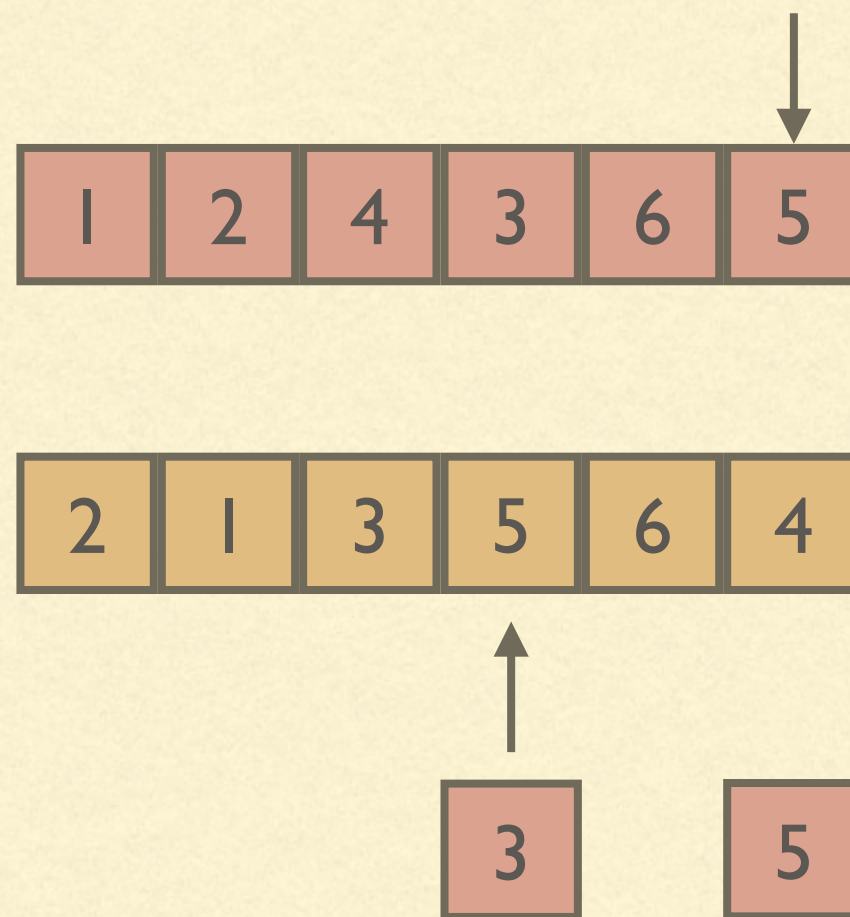
---



---

# CYCLE CROSSOVER

---



Once found we copy the value from the first parent.  
Repeat until we return to the beginning

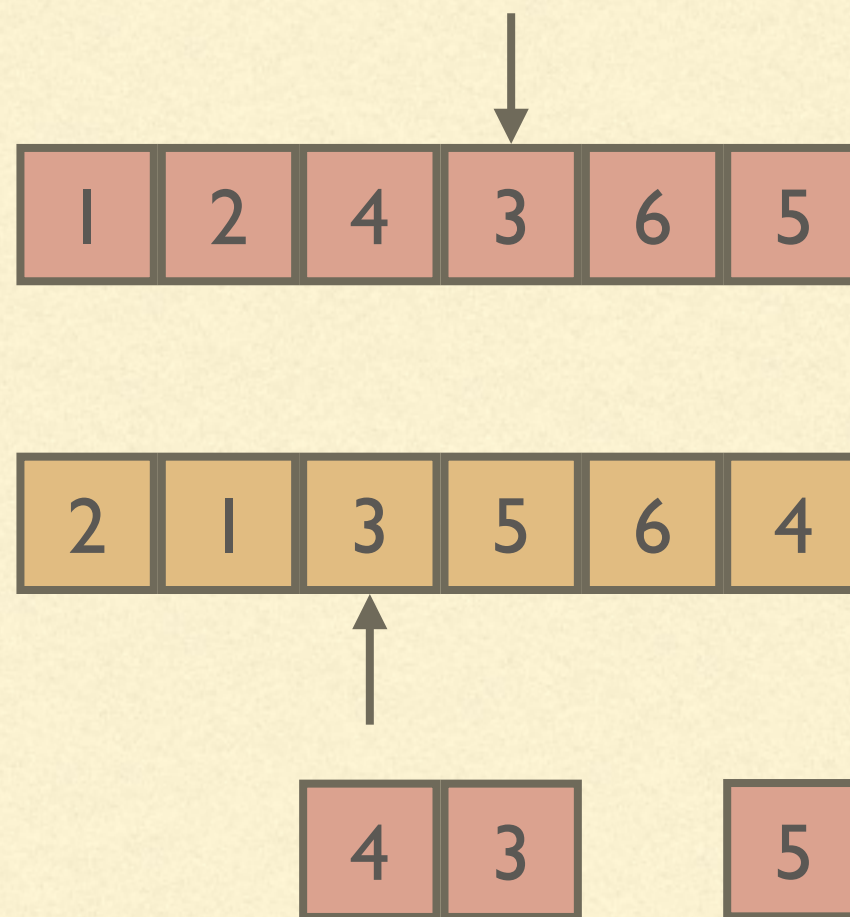
---



---

# CYCLE CROSSOVER

---



Once found we copy the value from the first parent.  
Repeat until we return to the beginning

---



---

# CYCLE CROSSOVER

---



We copy the remaining elements from the second parent

---



---

# REPRESENTING GRAPHS

---

- You might want to represent graphs. Possibly because they are ubiquitous in computer science.
  - You can represent graph in two ways:
    - *Direct encoding.* By actually representing vertices and edges
    - *Indirect encoding.* By representing some “device” that builds a graph
-

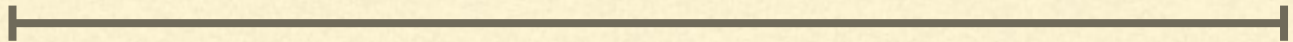


---

# ADJACENCY MATRIX

---

Side of the matrix = max number of nodes



0.4	no edge	0.6	-5.3
no edge	5.6	0.1	0.2
2.4	0.8	4.1	8.3
-0.2	no edge	0.5	no edge

Special value to represent  
missing edges

---



---

# LARGE GRAPHS

---

$V = \{a, b, c, d, e\}$       We evolve the sets of vertices and edges

$E = \{(a, b), (a, c), (d, a), (e, e)\}$

Possible mutations:

- Add an edge
- Add a node
- Remove an edge
- Remove a node and all its edges
- ...

Each one can have  
a different probability

Crossover is difficult  
to define and you might  
decide not to use it



---

# PRODUCTION RULES

---

- There are terminal symbols and non-terminal symbols
  - Production rules map a non-terminal symbol into a sequence/matrix of non-terminal and terminal symbols
  - We continue the expansion until the configuration is composed only of terminal symbols
  - By using adequate production rules we can encode indirectly a graph (i.e., rules that build a graph)
-



---

# PRODUCTION RULES

---

$$S \rightarrow \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

$$A \rightarrow \begin{bmatrix} c & p \\ a & c \end{bmatrix} \quad B \rightarrow \begin{bmatrix} a & a \\ a & e \end{bmatrix} \quad C \rightarrow \begin{bmatrix} a & a \\ a & a \end{bmatrix} \quad D \rightarrow \begin{bmatrix} a & a \\ a & b \end{bmatrix}$$

$$a \rightarrow \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad b \rightarrow \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \quad c \rightarrow \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \quad e \rightarrow \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \quad p \rightarrow \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

*Ruleset used in an example in:*

*Hiroaki Kitano, Designing neural networks using a genetic algorithm with a graph generation system*

---

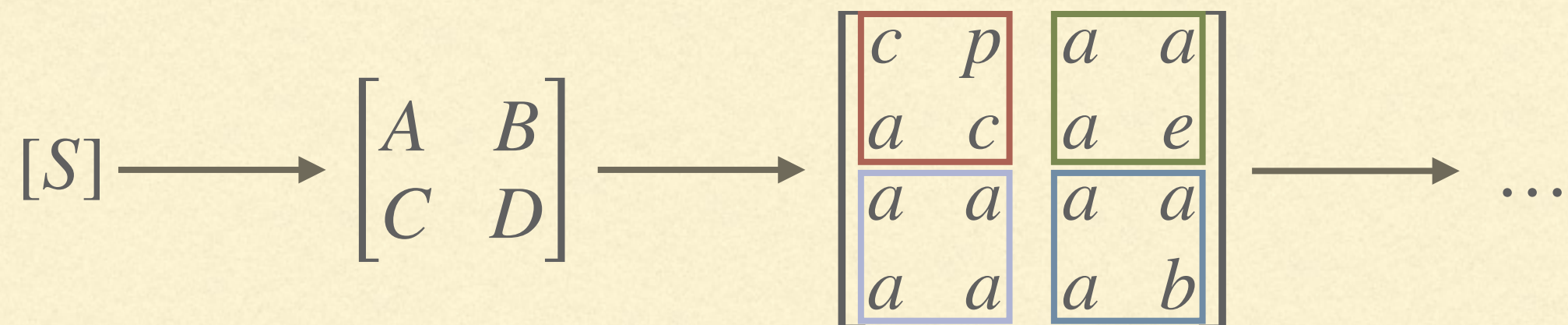


---

# PRODUCTION RULES

---

Starting from an axiom  $[S]$  we can iterate the production rules





# PRODUCTION RULES

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

A graph of 5 vertices  
(8 encoded but 3 of them  
are not connected to anything)



---

# PRODUCTION RULE: ENCODING

---



Since now we have a vector of fixed length,  
to perform the evolution  
we can apply traditional GA operators

---