CIS
RESEARCH GROUP

Heuristics & Metaheuristics for Optimization & Learning

Mario Pavone

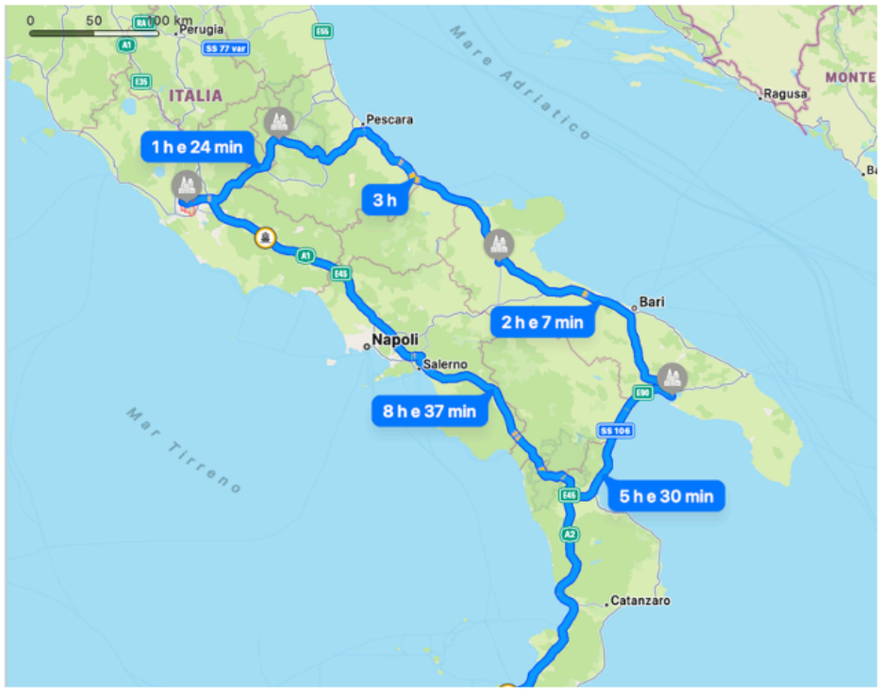
mpavone@dmi.unict.it

http://www.dmi.unict.it/mpavone/

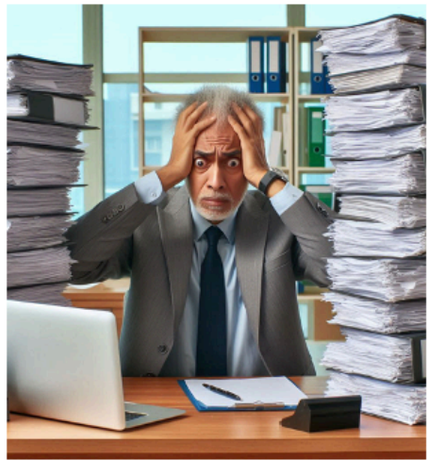
AA. 2024/2025 - LM 18

Why a problem is Hard?

Time



Uncertainty



Big Data

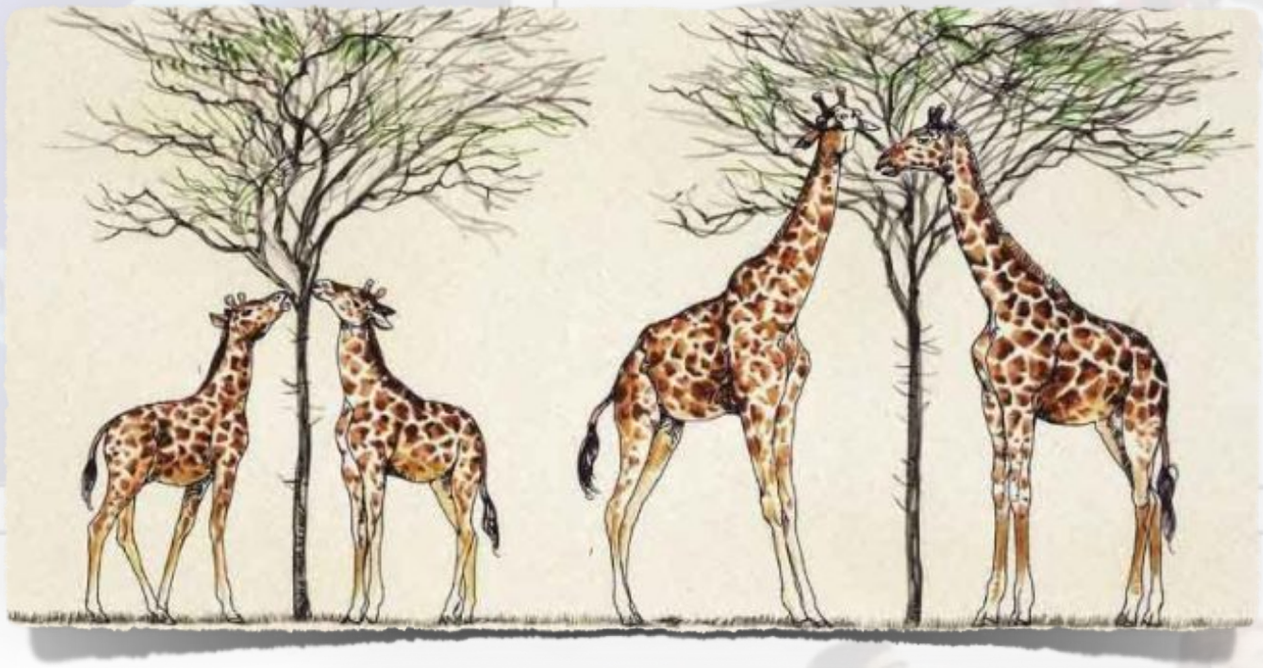
E ADESSO COSA FACCIO?

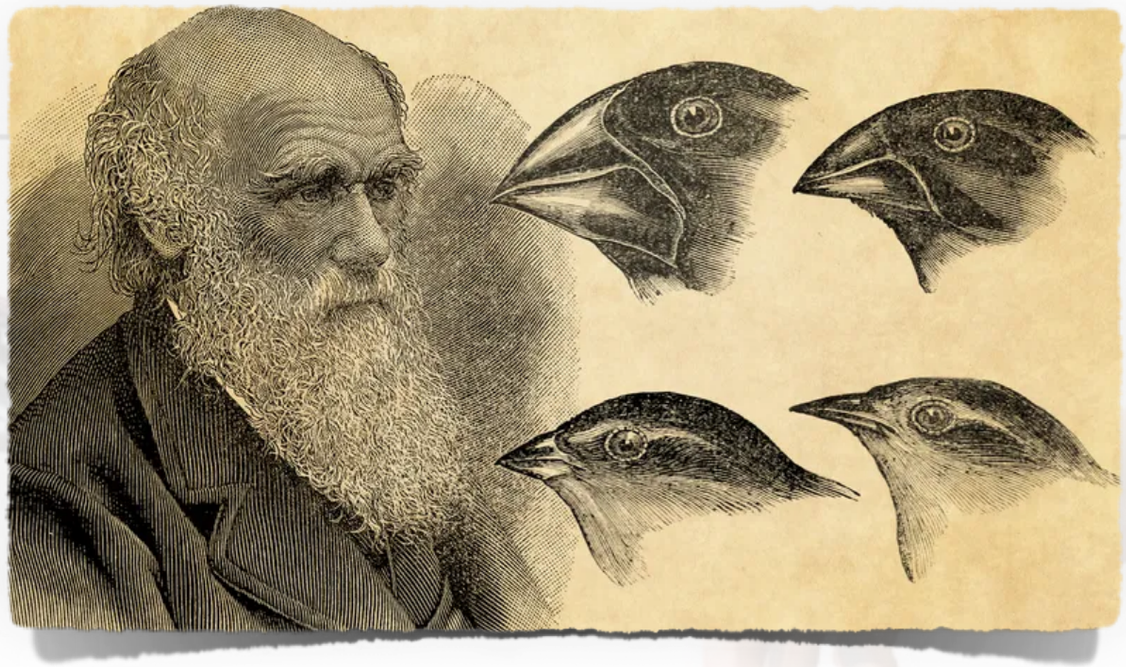
Look the nature and be inspired!

Population-Based Algorithms

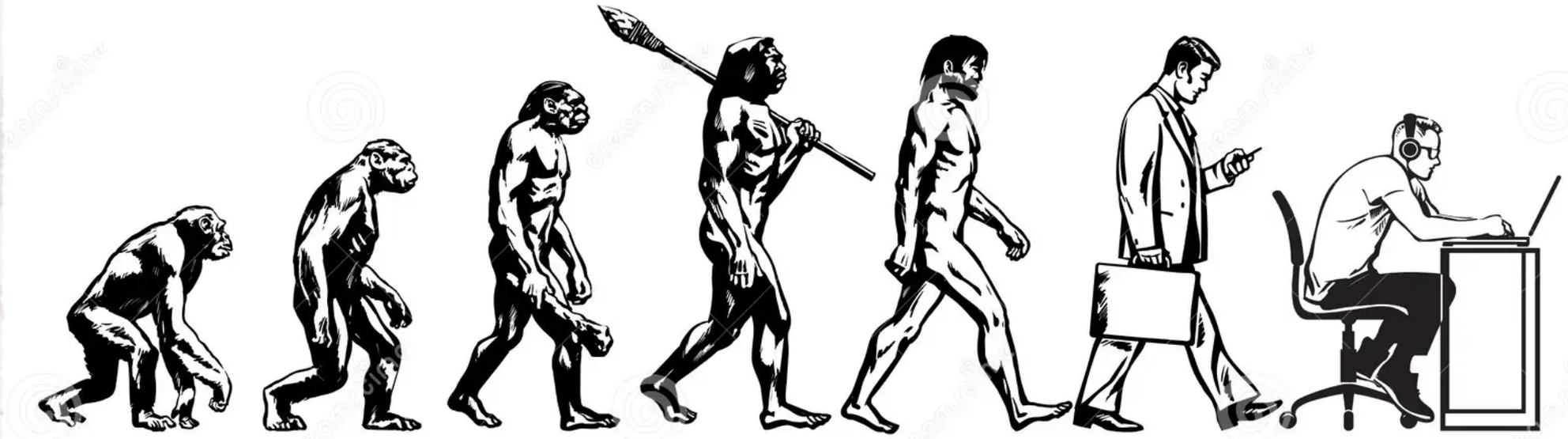
Look the nature and be inspired!


HOW

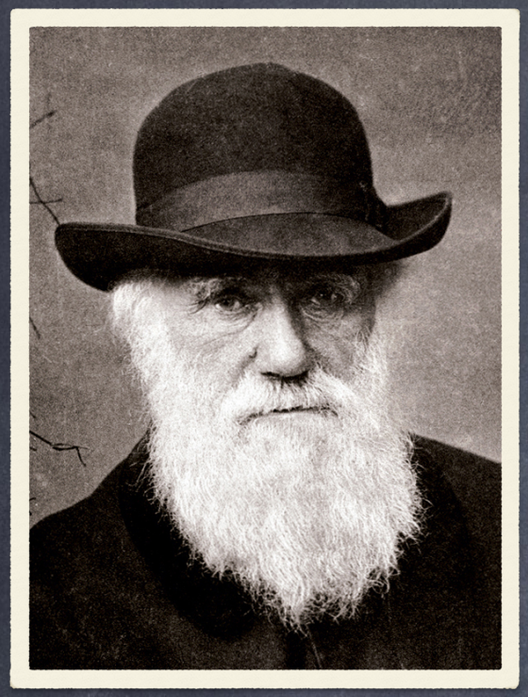




Teoria dell'Evoluzione



Teoria dell'Evoluzione



Charles Darwin

Selezione Naturale del *migliore*

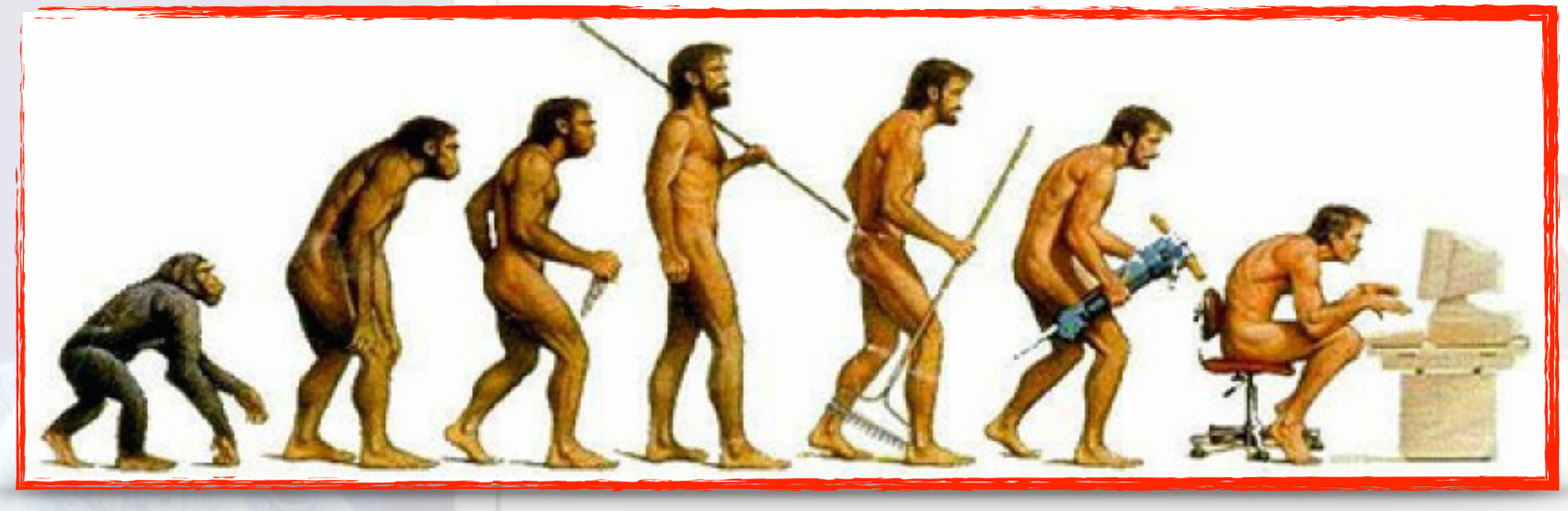
Adattamento

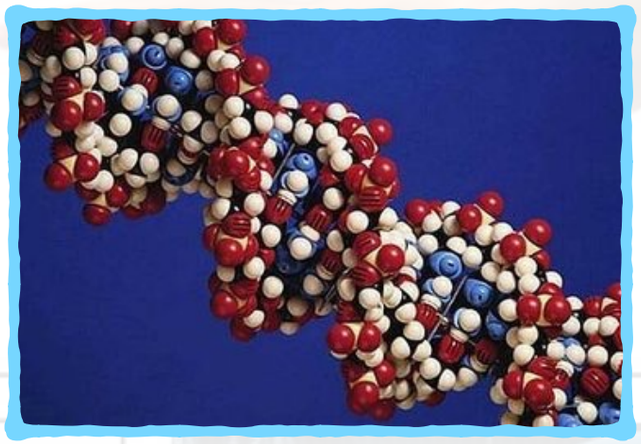
Crossover

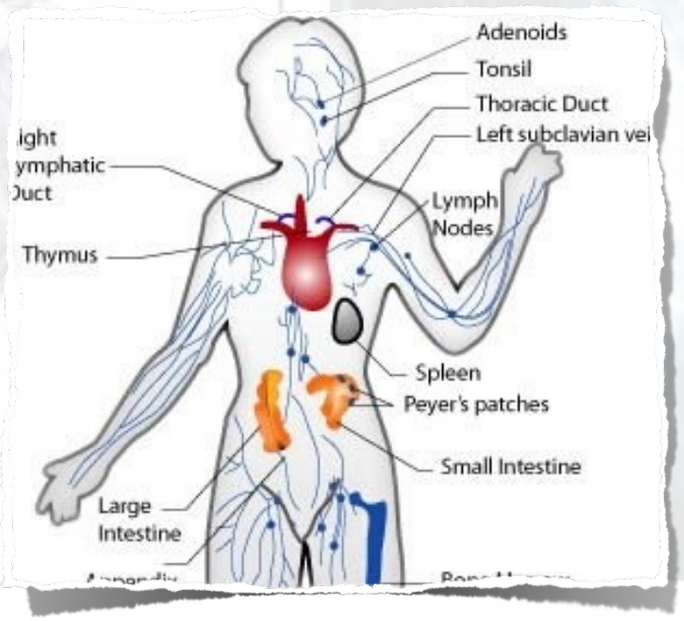
Mutazione

Evoluzione

Algoritmi Bio- & Nature-Inspired

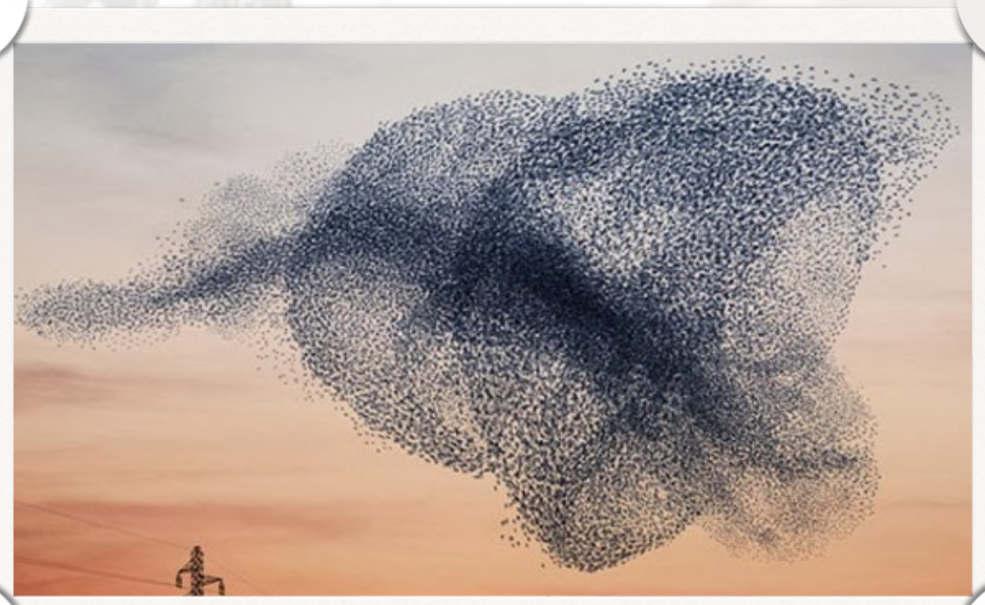


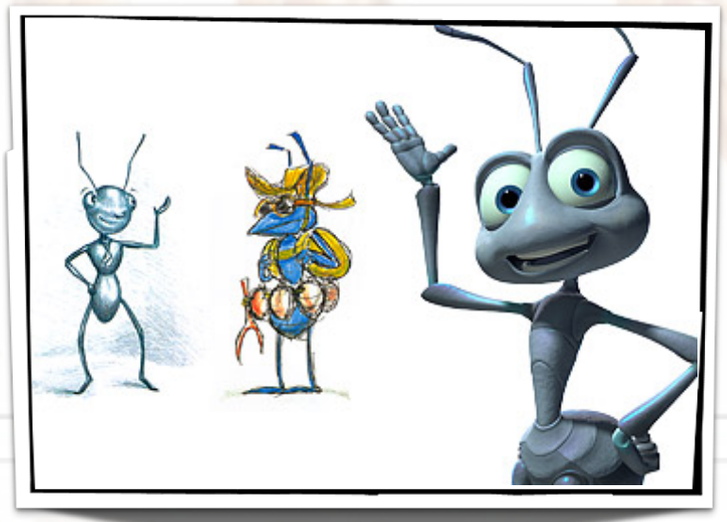


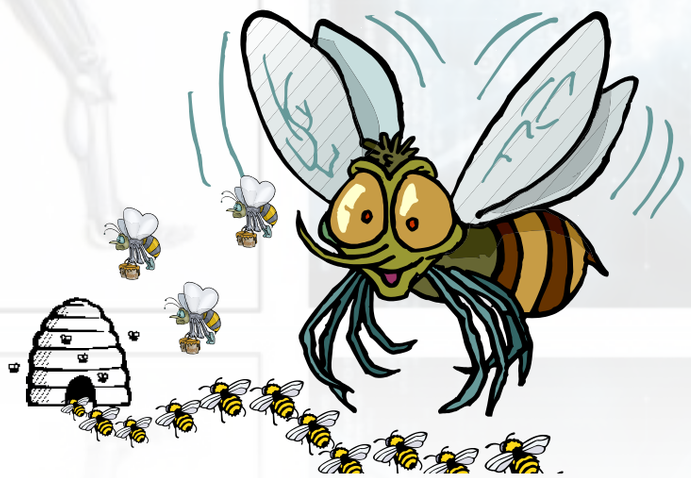


Swarm Intelligence

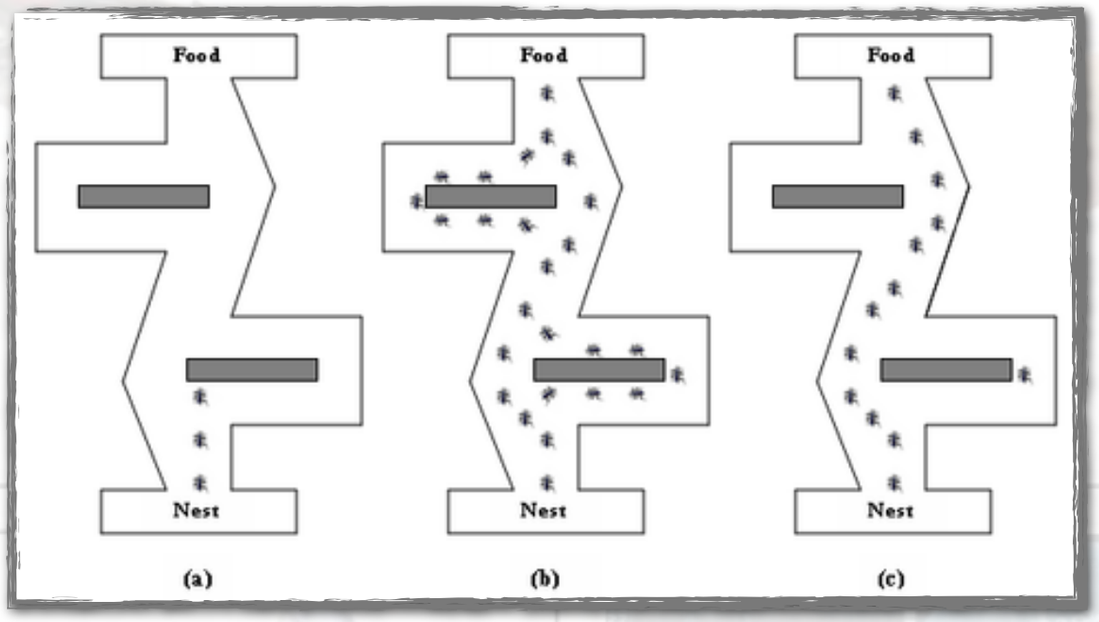
Comportamento Collettivo, Decentralizzato & Self-Adaptive

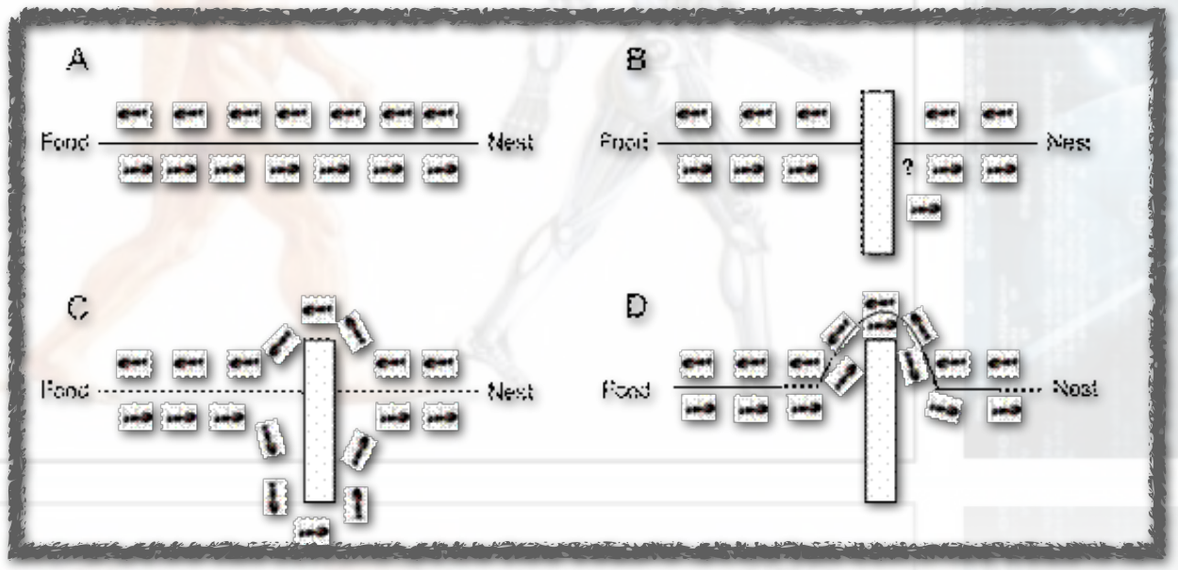






Colonie di Formiche





Computazione Naturale: i Vantaggi

Robustezza: in grado di affrontare ambienti con incertezza

Decentrata: senza un autorità centrale

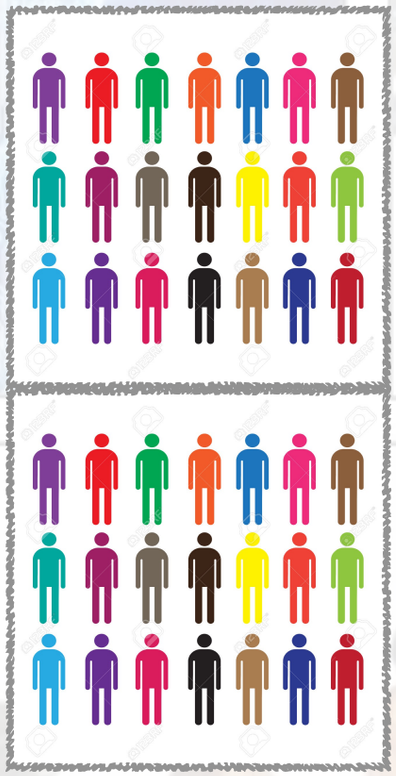
Autonomi: possono funzionare senza l'intervento dell'utente

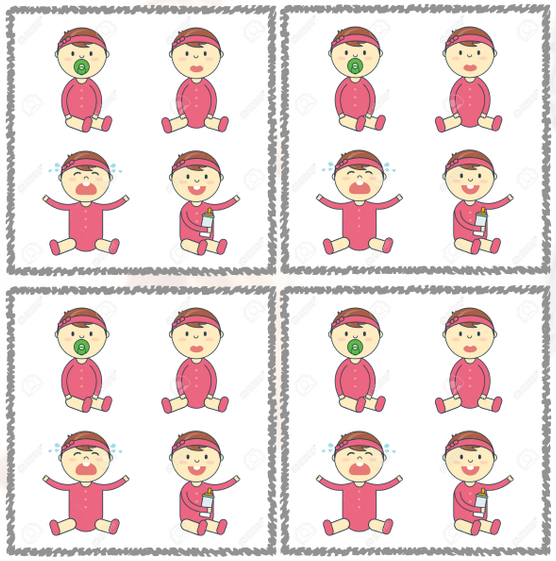
Flessibilità: applicabile a differenti problemi

Adattivi: in grado di gestire applicazioni in ambienti dinamici attraverso l'auto-adattamento

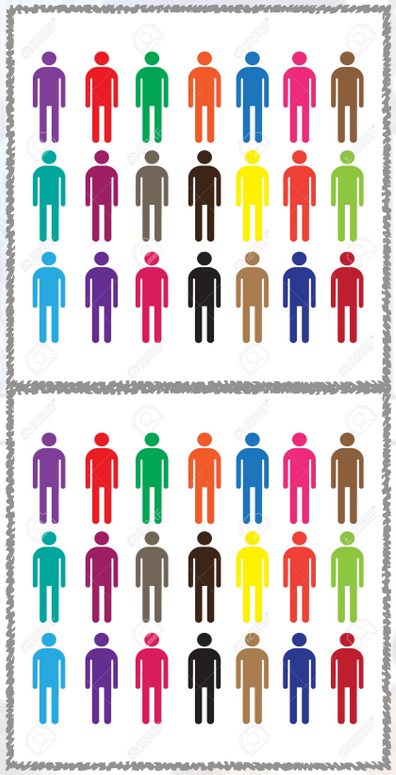
Come funzionano?

Evoluzione Naturale

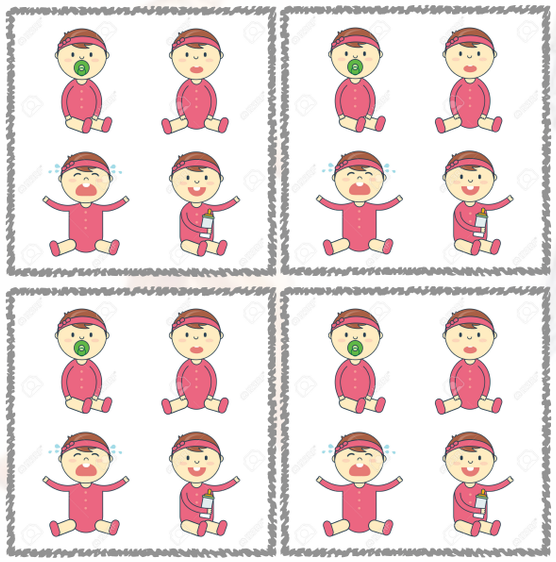




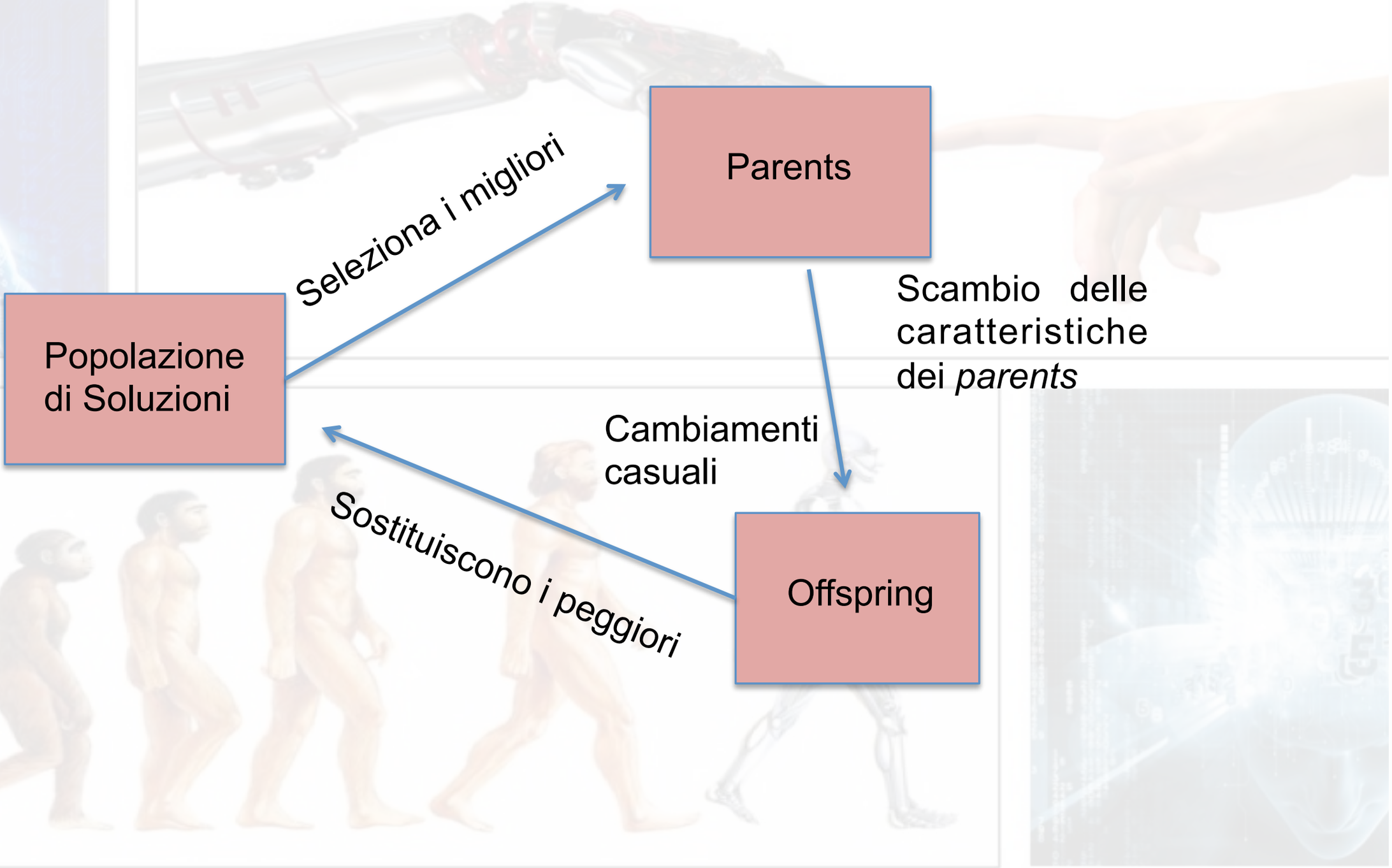
Evoluzione Naturale



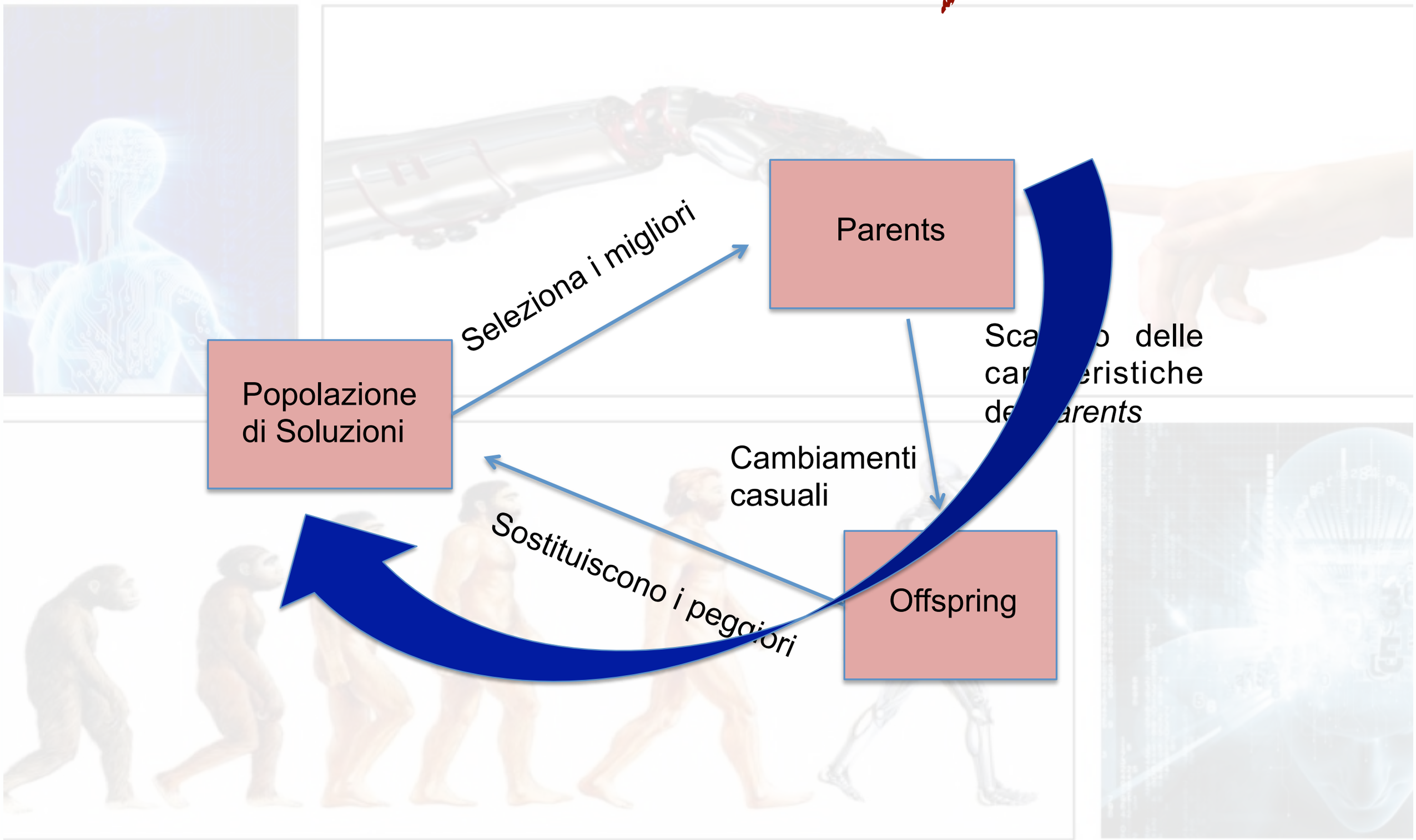
LOOP



Evoluzione Artificiale



Evoluzione Artificiale



A Genetic Algorithm: how it works

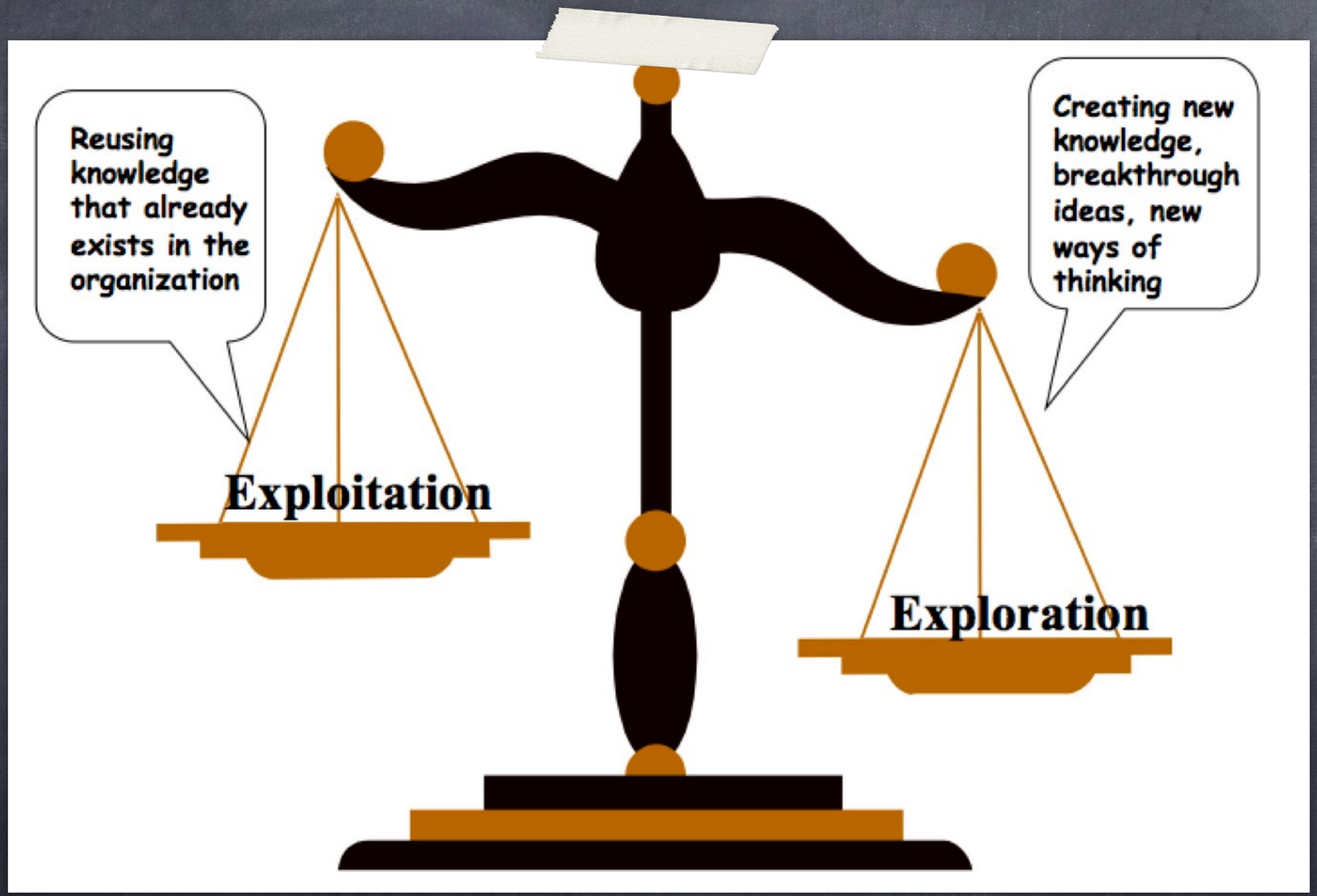
- Randomly generate a set of possible solutions to a problem, representing each as a fixed length character string
- Test each possible solution against the problem using a fitness function to evaluate each solution
- Keep the best solutions, and use them to generate new possible solutions
- Repeat the previous two steps until either an acceptable solution is found, or until the algorithm has iterated through a given number of cycles (generations)

Genetic Algorithm pseudocode

1. Generate the initial **population** $P(0)$ at random, and set $i \leftarrow 0$;
2. REPEAT
   (a) Evaluate the fitness of each individual in $P(i)$;
   (b) **Select** parents from $P(i)$ based on their fitness in $P(i)$;
   (c) **Generate** offspring from the parents using *crossover* and *mutation* to form $P(i+1)$;
   (d) $i \leftarrow i+1$;
3. UNTIL halting criteria are satisfied

Key GA Operators

- **Reproduction:** It is usually the first operator applied on population. Chromosomes are selected from the population of parents to cross over and produce offspring. It is based on Darwin's evolution theory of "Survival of the fittest". Therefore, this operator is also known as '**Selection Operator**'.
- **Cross Over:** After reproduction phase, population is enriched with better individuals. It makes clones of good strings but does not create new ones. Cross over operator is applied to the mating pool with a hope that it would create better strings.
- **Mutation:** After cross over, the strings are subjected to mutation. Mutation of a bit involves flipping it, changing 0 to 1 and vice-versa.


Reusing knowledge that already exists in the organization
Creating new knowledge, breakthrough ideas, new ways of thinking
Exploitation
Exploration

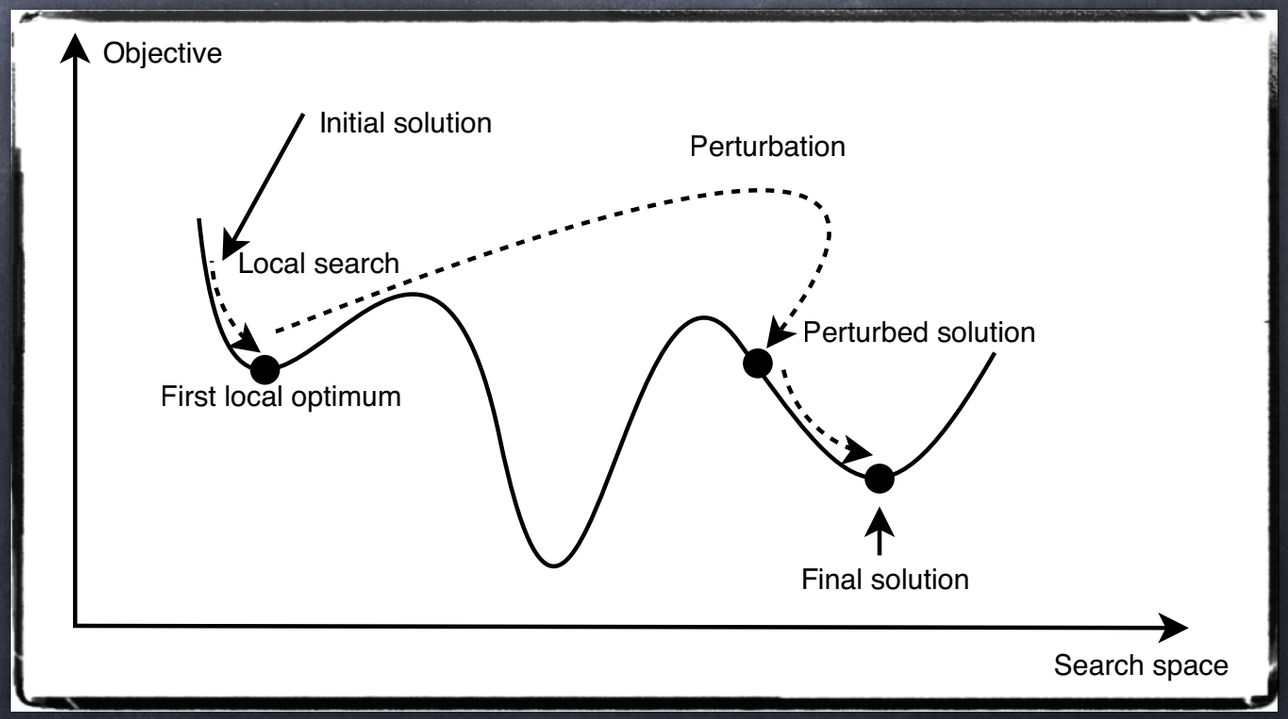
Chiave di Successione per un algoritmo Population-Based

Exploration of new parts of search space

- Mutation operators
- Recombination operators

Exploitation of promising genetic material

- Selection mechanism



Perturbation Operators

Crossover

x

Mutation

Schema di Selezione

Selection *emphasizes* the better solutions in a population

- One or more copies of good solutions.
- Inferior solutions are much less likely to be selected.
- Not normally considered a search operator,
  but influences search significantly

Selection and Reproduction

Selection *emphasizes* the better solutions in a population

- One or more copies of good solutions.
- Inferior solutions are much less likely to be selected.
- Not normally considered a search operator, but influences search significantly

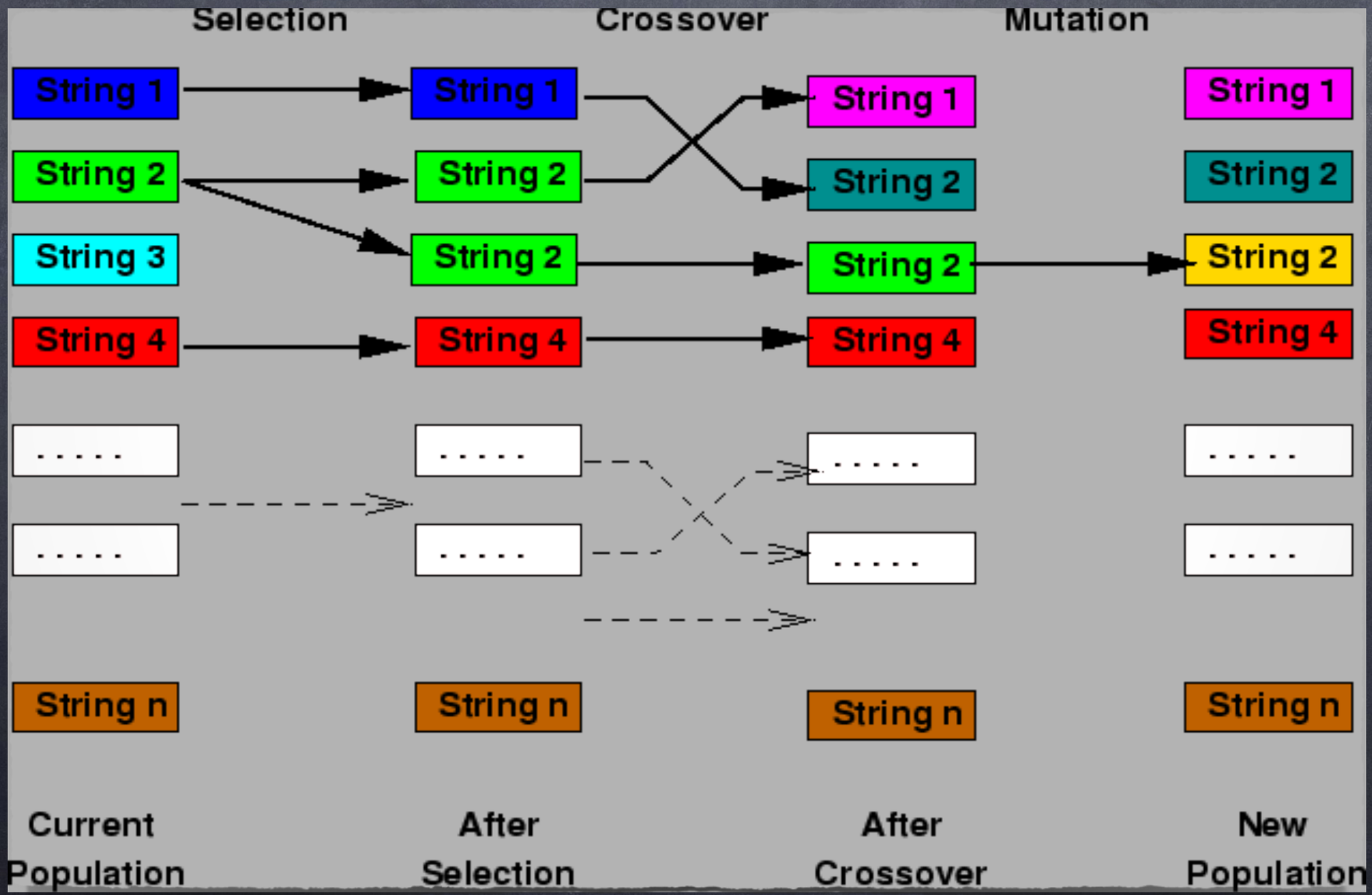
Selection can be used either before or after search operators.

- When selection is used before search operators, the process of choosing the next generation from the union of all parents and offspring is sometimes called *reproduction*.

Generational gap of EA

- refers to the overlap (i.e., individuals that did not go through any search operators) between the old and new generations.
- The two extremes are *generational* EAs and *steady-state* EAs.
- 1-elitism can be regarded as having a generational gap of 1.

The Basic Idea



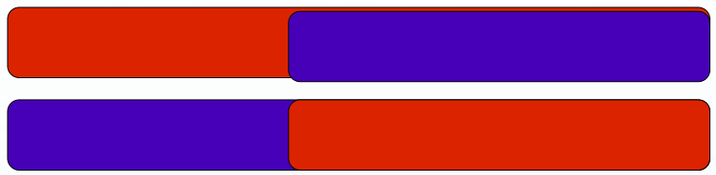
CrossOver Operator

- Once a pair of parent chromosomes is selected, the **crossover** operator is applied.
- First, the crossover operator randomly chooses a crossover point where two parent chromosomes "break" , and then exchanges the chromosome parts after that point. As a result, two new offspring are created.
- If a pair of chromosomes does not cross over, then the chromosome cloning takes place, and the offspring are created as exact copies of each parent.

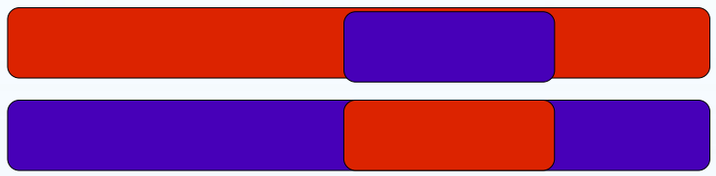
Kinds of CrossOver

- **Crossover**

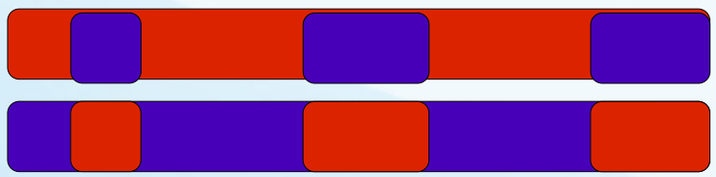
  - Single Point



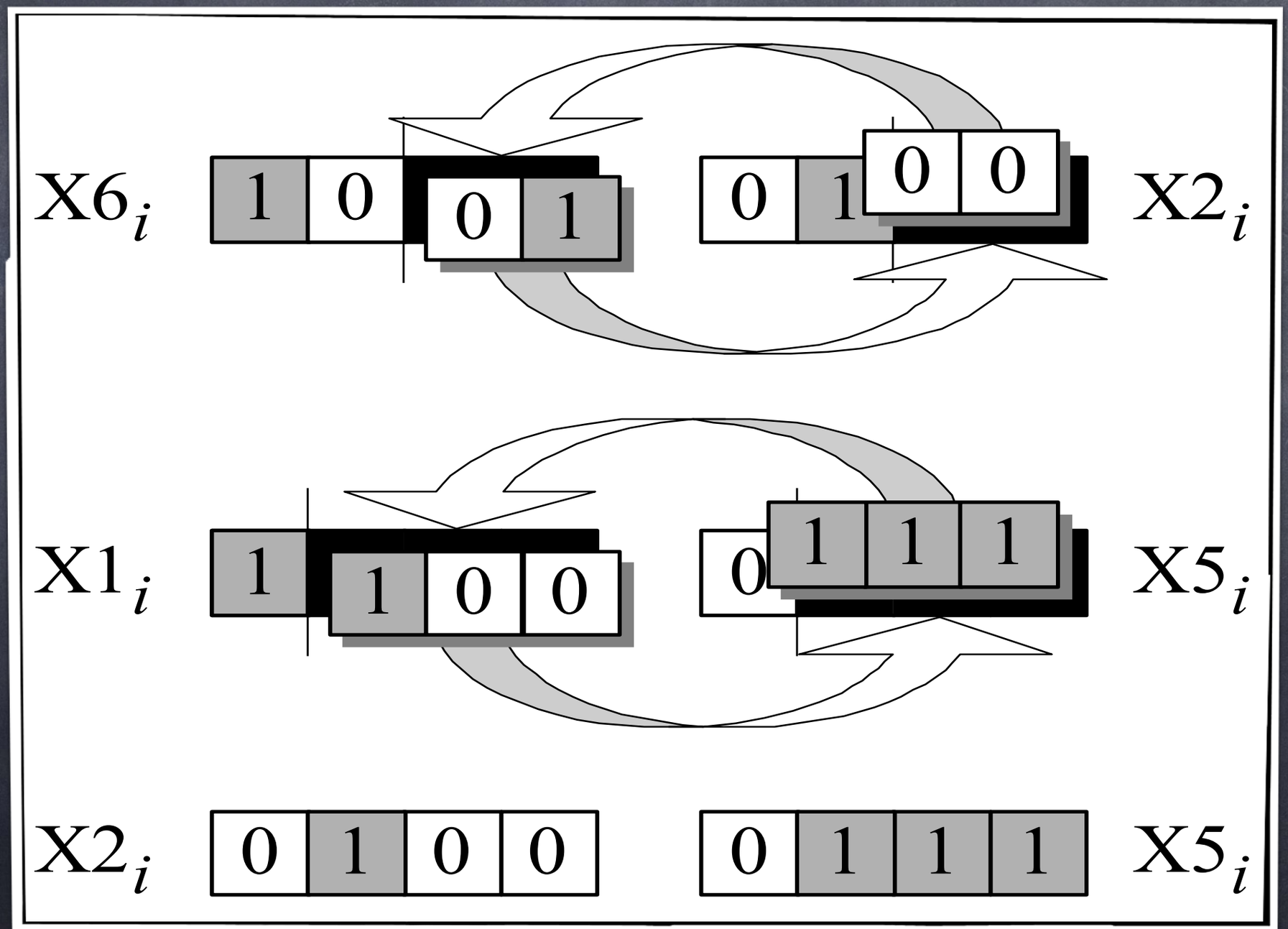
  - Two point



  - Uniform

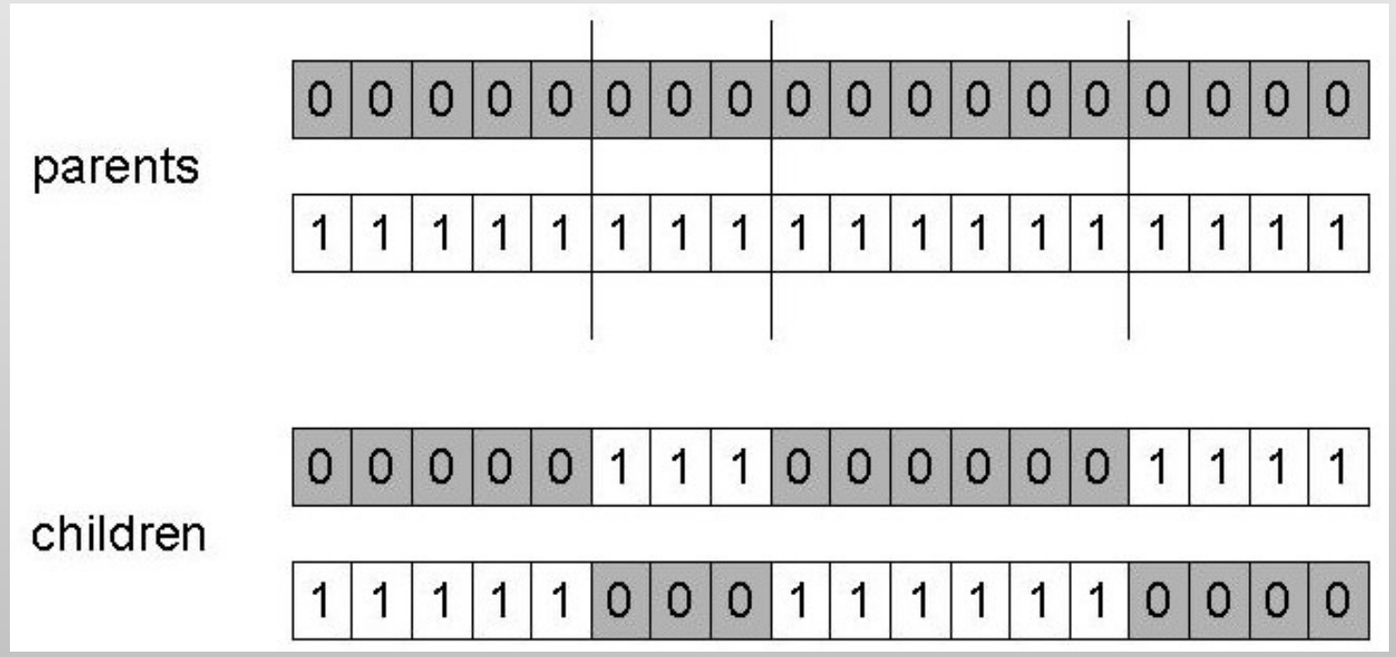


CrossOver: an example



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)



Uniform CrossOver

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

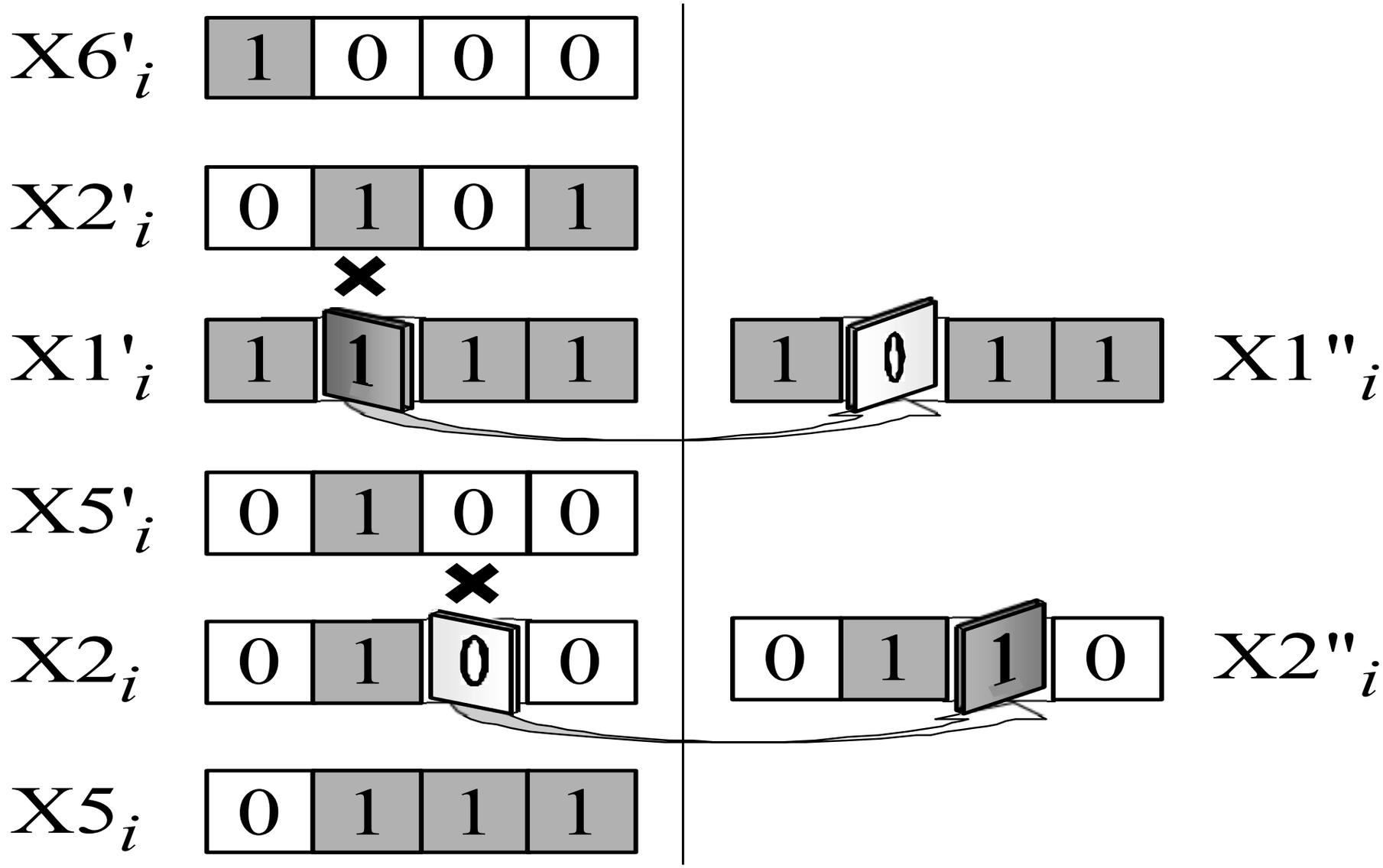
0	1	0	0	1	0	1	1	0	0	0	1	0	1	1	0	0	1

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

The Mutation Operator

- Mutation represents a change in the gene.
- Mutation is a background operator. Its role is to provide a guarantee that the search algorithm is not trapped on a local optimum.
- The mutation operator flips a randomly selected gene in a chromosome.
- The mutation probability is quite small in nature, and is kept low for GAs, typically in the range between 0.001 and 0.01.

Mutation Operator: an example



Other Kinds of Mutation Operator on order representation

- **Mutation**
  - Insert / Delete / Substitute
- Mass mutation -> harmful! (e.g. genetic disorder)
- Small changes -> beneficial! -> Variations!
- ***Help to better adapt to changes in their environment!***

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

Swap Mutation

- Pick two alleles at random and swap their positions
- Preserves most of adjacency information (4 links broken), disrupts order more

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

How does a simple GA work

Let's use the simple EA to maximise the function $f(x) = x^2$ with x in the *integer* interval $[0, 31]$, i.e., $x = 0, 1, \cdots, 30, 31$.

The first step of EA applications is *encoding* (i.e., the representation of chromosomes). We adopt binary representation for integers. Five bits are used to represent integers up to 31. Assume that the population size is 4.

1. Generate initial population at random, e.g., 01101, 11000, 01000, 10011. These are *chromosomes* or *genotypes*.
2. Calculate fitness value for each individual.
   (a) Decode the individual into an integer (called *phenotypes*),

$$01101 \to 13, 11000 \to 24, 01000 \to 8, 10011 \to 19;$$

How does a simple GA work

(b) Evaluate the fitness according to $f(x) = x^2$,

$$13 \to 169, 24 \to 576, 8 \to 64, 19 \to 361.$$

3. Select two individuals for crossover based on their fitness. If roulette-wheel selection is used, then

$$p_i = \frac{f_i}{\sum_j f_j}.$$

Two offspring are often produced and added to an intermediate population. Repeat this step until the intermediate population is filled. In our example,

$$p_1(13) = 169/1170 = 0.14 \quad p_2(24) = 576/1170 = 0.49$$
$$p_3(8) = 64/1170 = 0.06 \quad p_4(19) = 361.1170 = 0.31$$

Assume we have $crossover(01101, 11000)$ and $crossover(10011, 11000)$. We may obtain offspring 0110 0 and

How does a simple GA work

1100 1 from $crossover(01101, 11000)$ by choosing a random crossover point at 4, and obtain 10 000 and 11 011 from $crossover(10011, 11000)$ by choosing a random crossover point at 2. Now the intermediate population is

$$01100, 11001, 10000, 11011$$

4. Apply mutation to individuals in the intermediate population with a *small* probability. A simple mutation is bit-flipping. For example, we may have the following new population $P(1)$ after random mutation:

$$0110\mathbf{1}, 11001, \mathbf{0}0000, 11011$$

5. Goto Step 2 if not stop.

Representation of a problem

- Each individual corresponds to a solution $\boldsymbol{x} = x_1 x_2 \ldots x_n$
- We can modify the solution by means of crossover and mutation

Discrete representations

- Each gene has a value taken from a finite set
- E.g., $x_i \in \{0, 1\}$ or $x_i \in \{4, 5, 6, 7, 8\}$

Real-valued (continuous) representations

- Each gene has a value taken from a continuous interval
- E.g., $x_i \in [-5, 5]$ or $x_i \in [0, 1]$

Continuous Representation

$$f_1(x) = \sum_{i=1}^{n} x_i^2$$

$$f_2(x) = \sum_{i=1}^{n} |x_i| + \prod_{i=1}^{n} |x_i|$$

$$f_5(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$$

$$f_9(x) = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10]$$

Continuous Representation

Discrete Recombination

Does not change actual (gene) values

- Very similar to the crossover operators on binary strings

Multi-point Recombination (1-point, 2-point, k-point crossover)

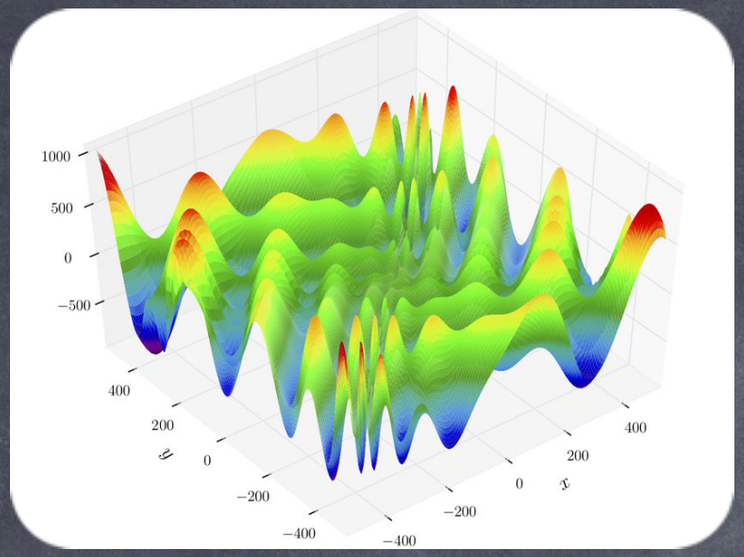
- Similar to that for the binary representation

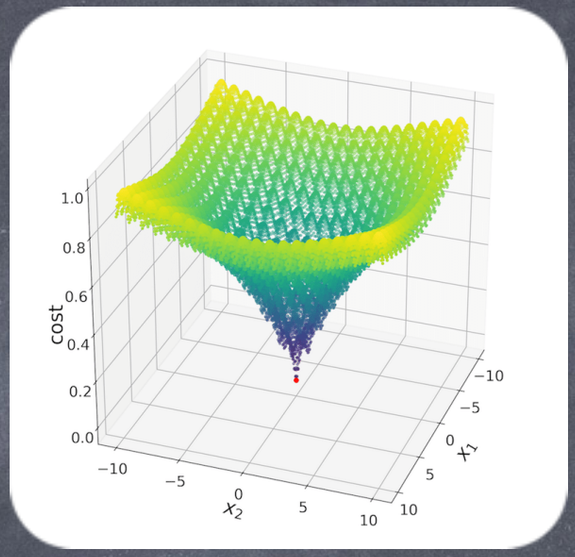
Global Discrete Recombination

- Similar to uniform crossover for the binary representation

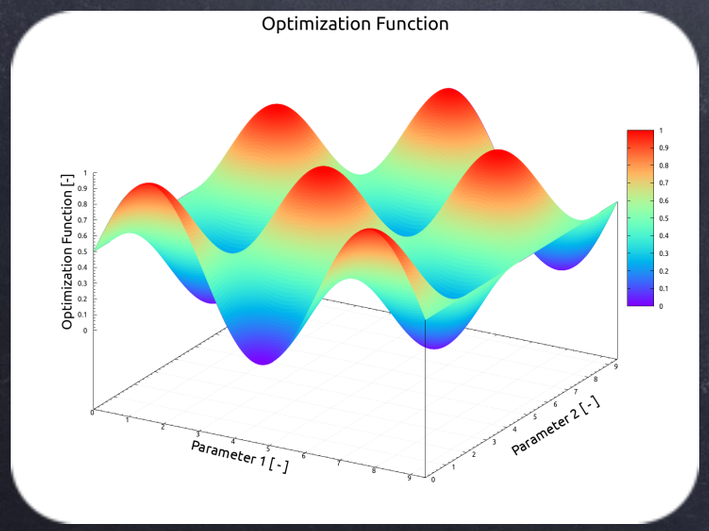
$\boldsymbol{x}_1$: 0.21 1.87 3.66 | 1.11 2.25 $\quad$ $\boldsymbol{x}'_1$: 0.21 1.87 3.66 | 2.56 0.11

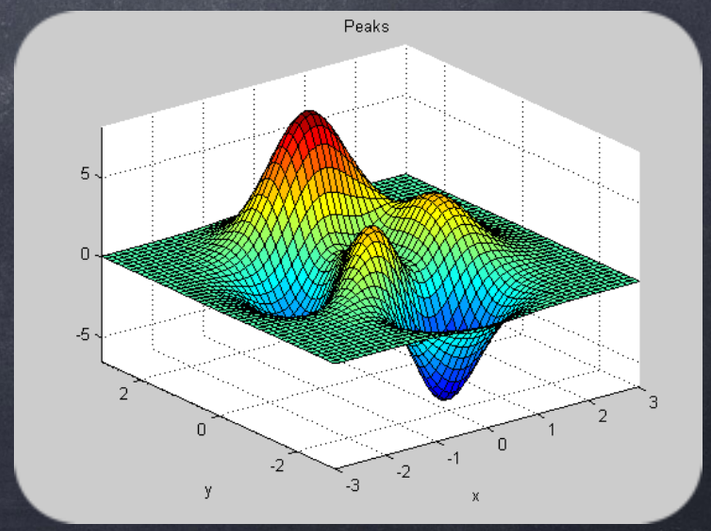
$\boldsymbol{x}_2$: 2.32 0.77 2.99 | 2.56 0.11 $\quad$ $\boldsymbol{x}'_2$: 2.32 0.77 2.99 | 1.11 2.25





Which mutation operators to use in Continuous Optimization?





Continuous Representation

Intermediate Recombination

For 2 parents $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ and $i = 1, 2, \ldots, n$:

$$x'_i = \alpha x_{1i} + (1 - \alpha) x_{2i} \quad \text{where} \quad \alpha \in [0, 1]$$

Given $\alpha = 0.5$:

$\boldsymbol{x}_1$: 0.21 1.87 3.66 1.11 2.25 $\boldsymbol{x}_2$: 2.32 0.77 2.99 2.56 0.11

$\boldsymbol{x}'$: 1.27 1.32 3.33 1.84 1.18

Arithmetic & Heuristic Recombination

Arithmetic crossover (for p parents):

$$x_i' = \alpha_1 x_{1i} + \alpha_2 x_{2i} + \alpha_3 x_{3i} + \ldots \quad \text{where} \quad \sum_{i=1}^{p} \alpha_i = 1$$

- Generalised (simple) intermediate crossover

Heuristic crossover (where $\boldsymbol{x}_1$ is no worse than $\boldsymbol{x}_2$):

$$\boldsymbol{x}' = u(\boldsymbol{x}_1 - \boldsymbol{x}_2) + \boldsymbol{x}_1 \quad \text{where} \quad u = rand([0, 1])$$

- Partially reflected point

Heuristic Recombination

Heuristic crossover example: Assume $f(\boldsymbol{x}_1) \geq f(\boldsymbol{x}_2)$ and $x_i \in [0, 4]$

$\boldsymbol{x}_1$: 0.21 1.87 3.66 1.11 2.25 $\qquad$ $\boldsymbol{x}_2$: 2.32 0.77 2.99 2.56 0.11

$$0.13 \cdot (0.21 - 2.32) + 0.21 \;,\; 0.47 \cdot (1.87 - 0.77) + 1.87 \quad \ldots$$

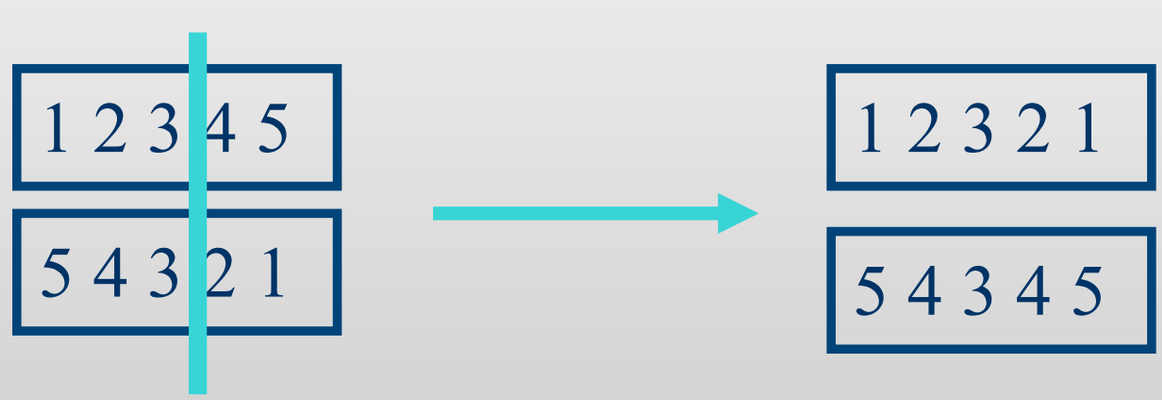
$$\boldsymbol{x}' : -0.06 \quad 2.39 \quad \ldots$$

What is happening here?

What if resulting values are out of bound (e.g., -0.06)?

Crossover For Permutations?

- "Normal" crossover operators will often lead to inadmissible solutions



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

Order Crossover Based

- Partially Matched Crossover (*PMX*)
- Order Crossover (*OX*)
- Cycle Crossover (*CX*)

- PMX
  - A = 9 8 4 | 5 6 7 | 1 3 2 10
  - B = 8 7 1 | 2 3 10 | 9 5 4 6

  - C = 9 8 4 | *2 3 10* | 1 *6 5 7*
  - D = 8 *10* 1 | *5 6 7* | 9 *2* 4 *3*

Partially Mapped Crossover (PMX)

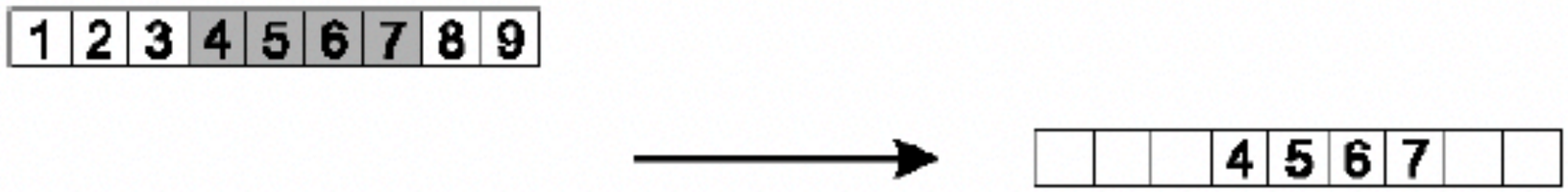
Informal procedure for parents P1 and P2:

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.

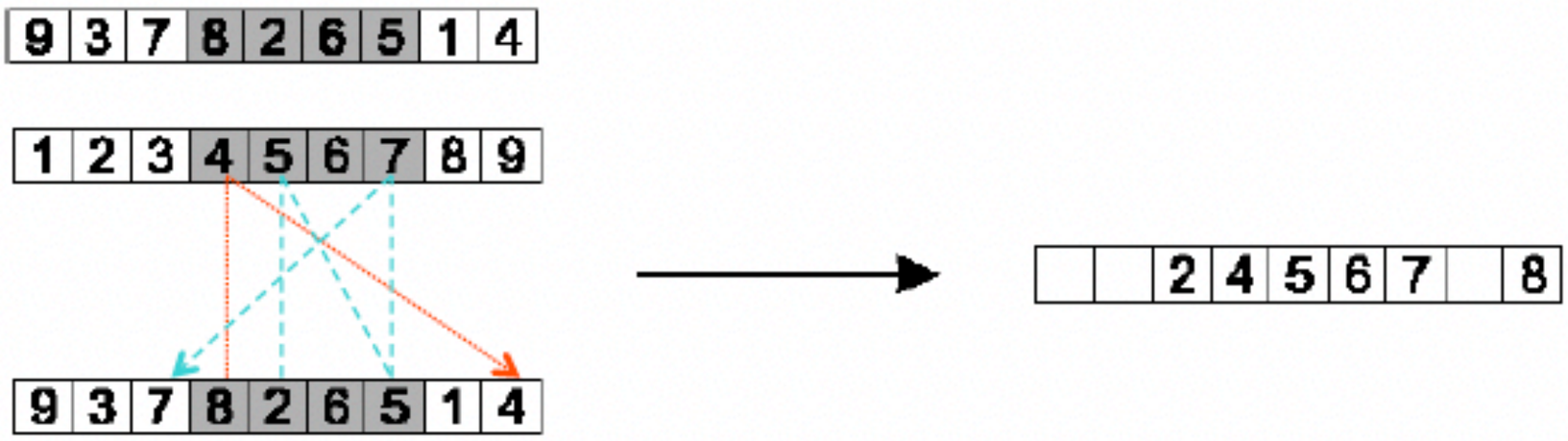
Second child is created analogously

PMX example

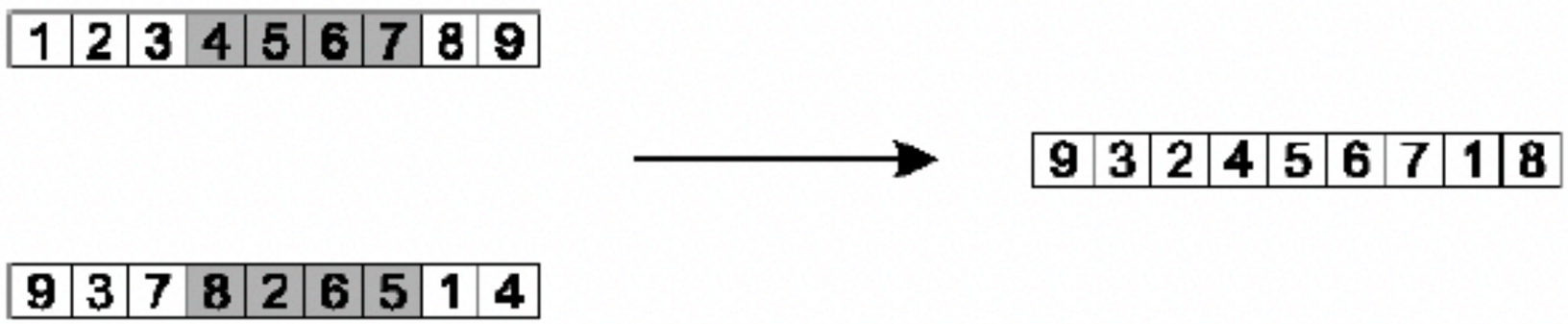
- Step 1



- Step 2



- Step 3



Order Crossover Based

- OX
  - A = 9 8 4 | 5 6 7 | 1 3 2 10
  - B = 8 7 1 | 2 3 10 | 9 5 4 6

  - C = 5 6 7 | 2 3 10 | 1 9 8 4
  - D = 2 3 10 | 5 6 7 | 9 4 8 1

- CX
  - A = 9 8 2 1 7 4 5 10 6 3
  - B = 1 2 3 4 5 6 7 8 9 10

  - C = 9 2 3 1 5 4 7 8 6 10
  - D = 1 8 2 4 7 6 5 10 9 3

Cycle crossover

Basic idea:

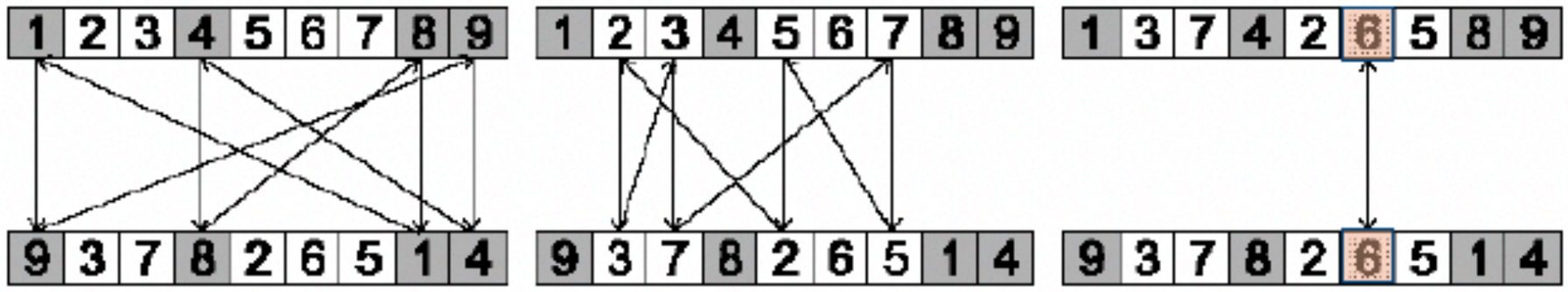
Each allele comes from one parent *together with its position*.

Informal procedure:

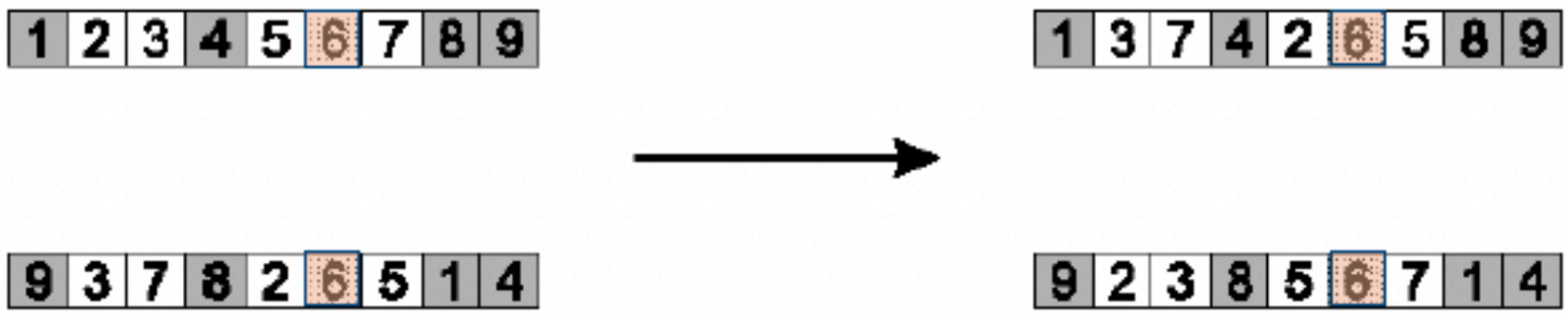
1. Make a cycle of alleles from P1 in the following way.
   (a) Start with the first allele of P1.
   (b) Look at the allele at the *same position* in P2.
   (c) Go to the position with the *same allele* in P1.
   (d) Add this allele to the cycle.
   (e) Repeat step b through d until you arrive at the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

Cycle crossover example

- Step 1: identify cycles



- Step 2: copy alternate cycles into offspring



Crossover OR mutation? (cont'd)

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

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Sudoku

- Il sudoku è un popolare rompicapo logico con i numeri. Le regole sono molto semplice. La griglia del sudoku è composta da 9x9 caselle divisa in altre griglie di 3x3 dette "regioni".

- Per risolvere il sudoku devi seguire queste direttive:
  - Ogni riga deve contenere tutti i numeri da 1 a 9.
  - Ogni colonna deve contenere tutti i numeri da 1 a 9.
  - Ogni regione deve contenere tutti i numeri da 1 a 9.

Key Concepts to Pay Attention

- Codifica del problema
- Rappresentazione della soluzione
- Progettazione metodo/processo di ricerca della soluzione
- Definizione della funzione obiettivo

Una soluzione può essere inizializzata casualmente o riempiendo alcune celle con indizi dati.

La funzione di fitness può essere definita come il numero di vincoli violati da una soluzione. Un vincolo è violato se una riga colonna o sottogriglia contiene duplicati. L'obiettivo è minimizzare il valore della funzione di fitness che è zero per una soluzione valida del sudoku.

Il vicinato di una soluzione può essere generato scambiando due celle all'interno di una riga colonna o sottogriglia. Questo assicura che il numero di celle riempite rimanga costante.

Tabella sudoku

Input:

3 0 6 5 0 8 4 0 0
5 2 0 0 0 0 0 0 0
0 8 7 0 0 0 0 3 1
0 0 3 0 1 0 0 8 0
9 0 0 8 6 3 0 0 5
0 5 0 0 9 0 6 0 0
1 3 0 0 0 0 2 5 0
0 0 0 0 0 0 0 7 4
0 0 5 2 0 6 3 0 0

Output:

3 1 6 5 7 8 4 9 2
5 2 9 1 3 4 7 6 8
4 8 7 6 2 9 5 3 1
2 6 3 4 1 5 9 8 7
9 7 4 8 6 3 1 2 5
8 5 1 7 9 2 6 4 3
1 3 8 9 4 7 2 5 6
6 9 2 3 5 1 8 7 4
7 4 5 2 8 6 3 1 9