

GENETIC ALGORITHMS

AN ITERATIVE WAY OF
IMPROVING A
POPULATION OF SAMPLE

A genetic algorithm is an optimization algorithm inspired by the theory of natural evolution. It relies on natural selection, reproduction, and mutation to generate a population of solutions that adapt and converge toward an optimal solution. Each individual is represented by a chromosome that contains its genes, which represent possible solutions. The best individuals are selected for reproduction, generating a new population of individuals. During the reproduction process, random mutations can occur to explore new solutions. The algorithm continues until a stopping condition is reached, such as the maximum number of generations or convergence toward an optimal solution.

GENETIC ALGORITHMS

- THE FIRST THOUGHT WAS BY VON NEUMANN
- 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
=> SURVIVE THROUGH AN ENVIRONMENT HOW? EVOLVING AND ADAPTING
- 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 works on the genotype and some on the phenotype

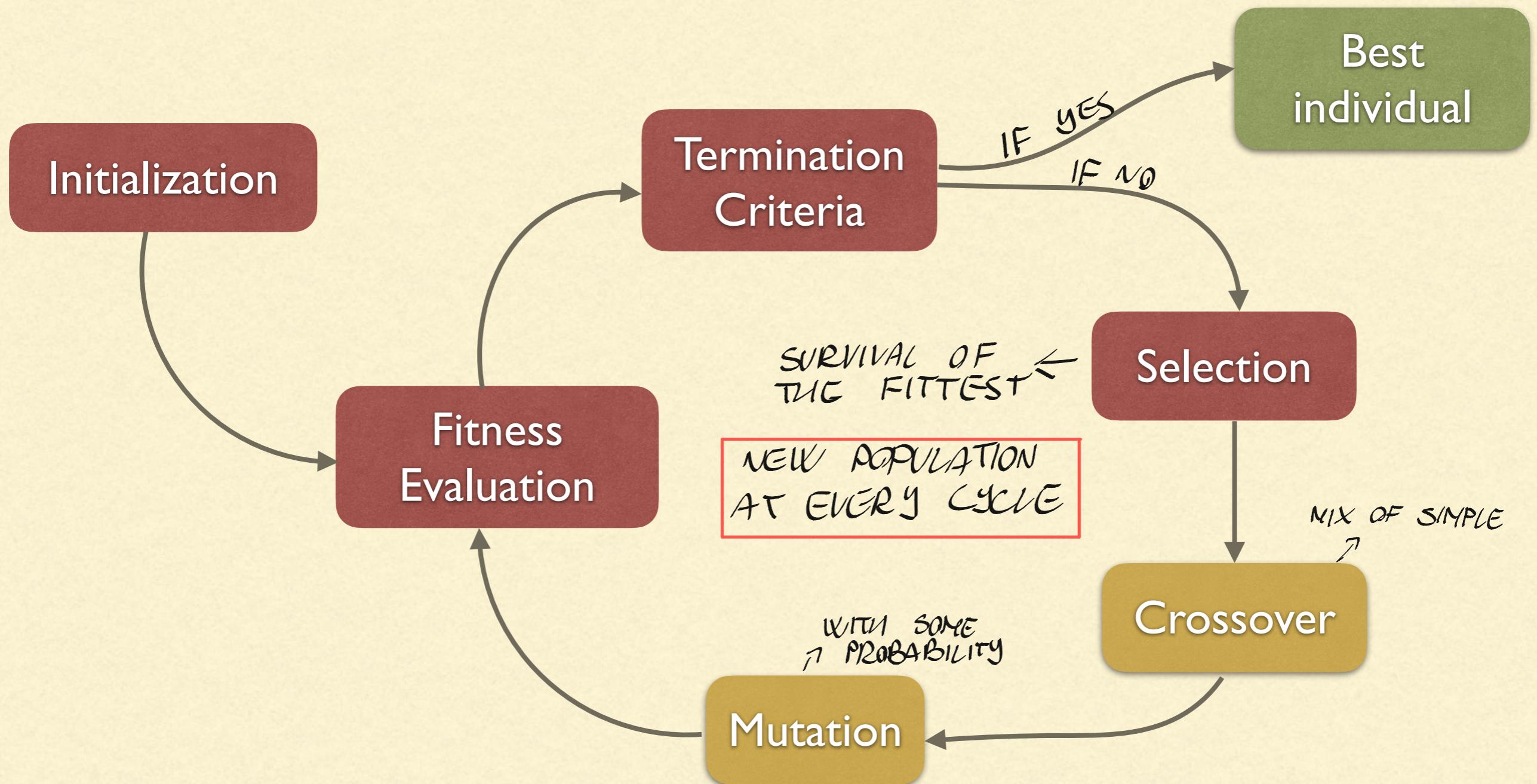
PHENOTYPE AND GENOTYPE

↑ IS THE DNA

- The **genotype** is how a solution is represented. (OR ENCODED)
E.g., a string of bits.

A GENOTYPE IS A SEQUENCE OF
FLOATING POINT NUMBER OR MATRIX
OR GRAPH ENCODED VIA MATRIX.
IS HOW A SOLUTION IS REPRESENTED
- Crossover and mutation works on the genotype of a solution
THEY NEED TO COMBINE OR CHANGE, SO THEY HAVE TO WORK ON THE SOLUTION
- The **phenotype** is the actual solution. E.g., the number of ones, the number represented by a string of bits, etc.
YOU CHANGE THE REPRESENTATION, YOU CHANGE THE PHENOTYPE
THE GENOTYPE IS NOT BOTHERED OF THE CHANGE
THE PHENOTYPE DON'T CHANGE
AND YOU DON'T CONSIDER THE SOLUTION
- 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



INITIALIZATION

IT'S LESS IMPORTANT FOR GA CAUSE WE USUALLY USE A NORMAL DISTRIBUTION UNLESS WE KNOW SOMETHING ABOUT THE DOMAIN OF THE SPACE.

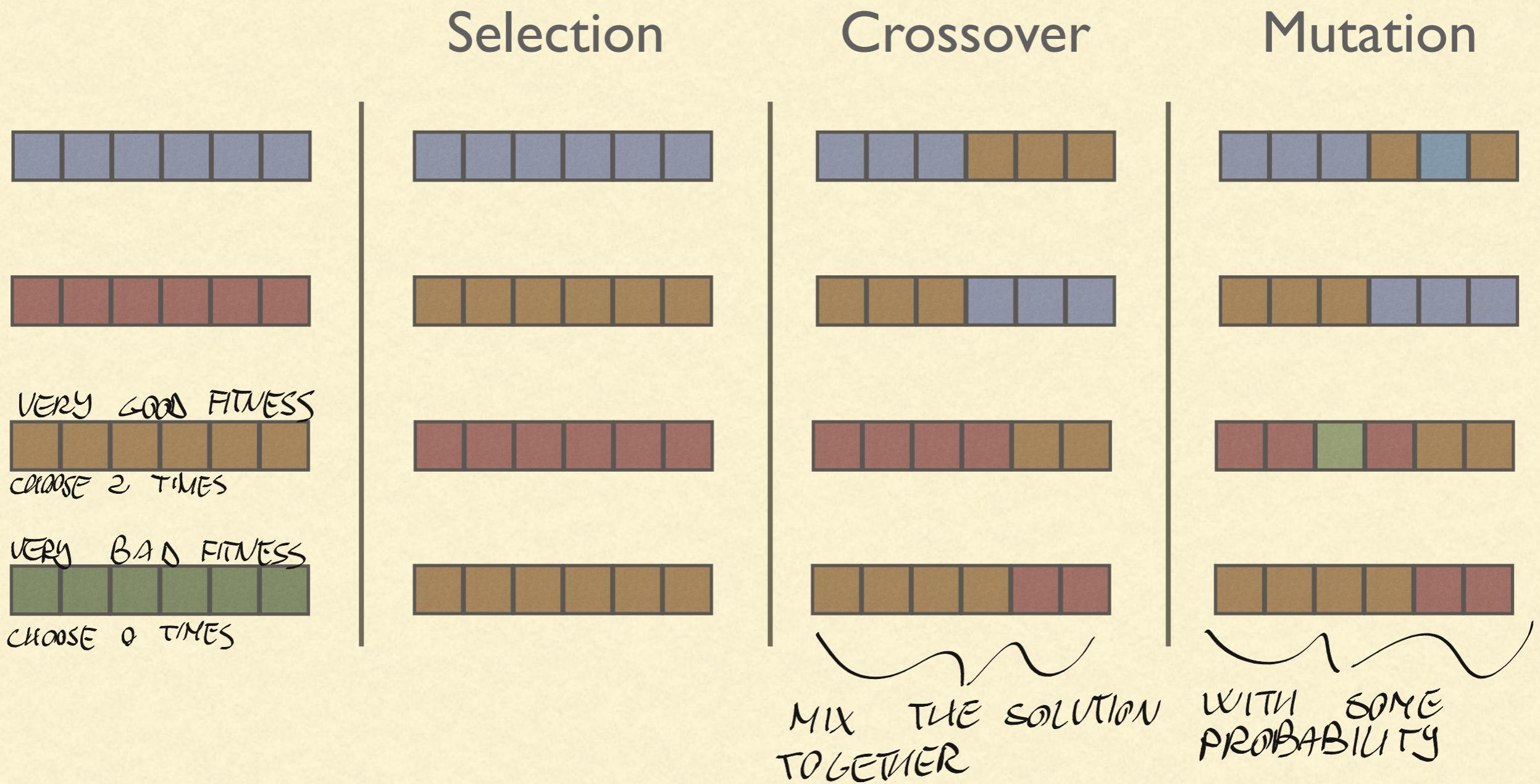
ONE OF THE POSSIBILITY IS CALLED CLEAVING:

IF YOU HAVE GOOD SOLUTIONS YOU CAN TRY TO GENERATE PART OF THE POPULATION AND ADD THE GOOD SOLUTIONS AS BOOST THE MISSION OF SEARCH.

THE PROBLEM IS THAT YOU RISK TO FIND SMALL VARIANTS OF THE SOLUTION THAT YOU ALREADY HAVE.
THEY ARE GOOD AND SO TEND TO DOMINATE THE OTHERS.

"IS LIKE PUTTING AN ANIMAL IN AN ENVIRONMENT WITHOUT A PREDATOR THAT HUNT IT"

A GA GENERATION



GA PARAMETERS

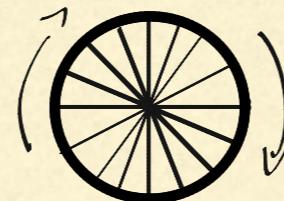
- Population size N $\sim 700 \rightarrow \sim 1000$ THE POPULATION HAVE TO BE VARIATED
IF TOO SIMILAR THEY ARE GONNA DO THE SAME THINGS
- Type of selection
- Type of crossover
- Crossover probability p_{cross} , usually near one
- Type of mutation
- Mutation probability p_{mut} , usually small
- Elitism or others additional modifications

SELECTION METHODS

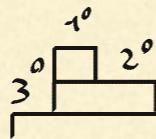
IT DEPENDS ALL ON THE POPULATION

- There are multiple ways to decide which individuals are selected:

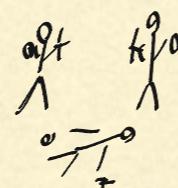
- Roulette wheel selection



- Ranked selection



- Tournament selection



ROULETTE WHEEL SELECTION

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

FUNCTION TO MAXIMISE

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

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

Ex: IF ALL INDIVIDUAL HAVE 1000 AS
FIT HE'S ALWAYS GONNA BE CHOSEN

SOLUTION

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^{th} best fitness will be selected with probability $\frac{N - i + 1}{r_{\text{sum}}}$ where r_{sum} is a normalisation factor
WE LIKE NORMALIZED VALUES
 - WE STILL NEED A FUNCTION THAT CONVERT RANK TO PROBABILITY
WE NEED TO BUILD A FAMILY OF FUNCTIONS

TOURNAMENT SELECTION

A BATTLE ROYAL OF SOLUTIONS WHERE THE FITTEST SURVIVE

- 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). THANKS TO REINSERTION WORTH INDIVIDUAL ARE ABLE TO SURVIVE.
- 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

MORE COMPETITION

- Large tournament size → high selection pressure

LESS COMPETITION

- Low tournament size → low selection pressure

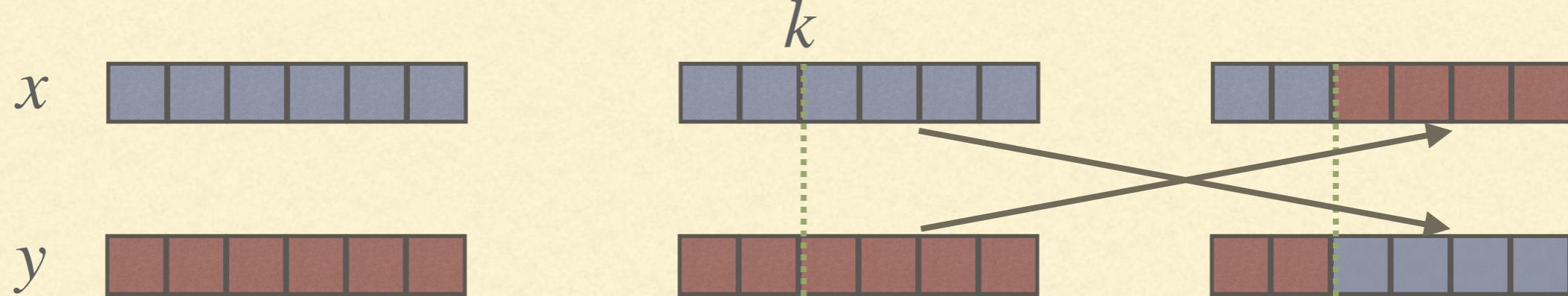
THE TOURNAMENT SIZE IS AN HYPERPARAMETER THAT HAVE TO BE TUNED. IT DEPENDS A LOT ON THE SIZE OF THE POPULATION. SOMETIMES IT CHANGE DURING THE EXECUTION OF THE CYCLE

Crossovers

MIX SOLUTIONS: · GOOD ONE
· VERY GOOD ONE > POTENTIALLY THE OPTIMUM ONE

- 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
WE ARE NOT INTERESTED IN THE Fittest
- The two offsprings are $x_1\cdots x_k y_{k+1} \cdots y_n$ and $y_1 \cdots y_k x_{k+1} \cdots x_n$

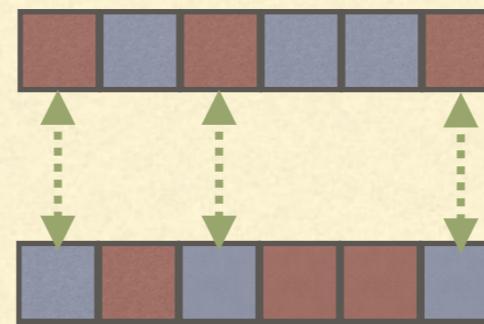
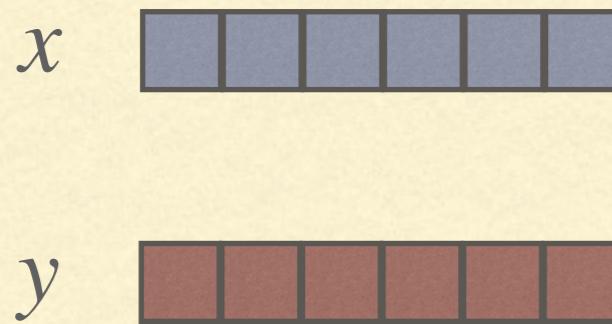
m-POINTS CROSSOVER

- 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

MOST INTERSTING:

- MOST RANDOMIC
- BEST PERFORMING



On average half of the bits will be exchanged

- 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^{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

x 



- 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_{mut}
- Usually $p_{\text{mut}} = \frac{1}{n}$ to mutate (on average) one bit per individual

Crossover and Mutation

MIXING BIT

FLIPPING BIT

↑ THE BIT IS JUST
AN EXAMPLE, IT COULD
BE ANYTHING ELSE

- 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

Ex: WE REPLACE THE ~10% OF THE POPULATION

- Instead of creating a completely new population at each iteration we only replace some of the individuals in the existing population
↳ BY OTHER INDIVIDUALS. WE ARE NOT REMOVING
- Which individuals to replace?
 - The worst ones
 - A random selection

EACH INDIVIDUAL HAVE A
CHANCE OF REMAINING IN
THE POPULATION

HYBRID GA

LOCAL SEARCH + GA

WE MIX THE CONCEPT OF SEARCHING
THE BESTS INDIVIDUAL WITH EXPLORING
THE ENVIRONMENT FOR THE BEST
SOLUTION.

... → CROSS OVER → MUTATION → LOCAL SEARCH → NEW GEN → ...

- After each generation all individuals are improved with a local search algorithm (e.g., hill climbing)
IT'S LIKE THE POPULATION "TRAIN" A LITTLE BIT
- Called “Memetic Algorithms”, better names are “Lamarckian algorithms” or “Baldwin Effect Algorithms”
- Parameters:
 - WHICH LOCAL SEARCH TO USE
 - How frequently the local search is performed
 - How many iterations of local search are performed each time

IT APPLY WELL IN NEURAL NETWORK: I DON'T WANT TO OPTIMIZE WITH
GRADIENT DESCENT, SO WE COMBINE THE TWO

REPRESENTATION

- **Beyond Binary.** It is possible to generalize GA to use symbols from a finite set/alphabet Σ instead of using only $\{0,1\}$
 - THE SELECTION DON'T CARE ABOUT THE TYPE. THE CYCLE IS THE SAME.
 - 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 Σ (e.g., $\Sigma = \{0,1,2,3\}$) then mutation can be changed to be, for example “add or subtract one”.
- $\Sigma = \{A, B, C, D, \dots, Z\}$
SAMPLE = A F G H I O Z $\xrightarrow{\text{MUTATION}}$ B F G I I P Z

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...

ONE SMALL PROBLEM:

- ...but this means that different bits have different impact on the encoded number

MAYBE YOU ARE CHANGING A LESS IMPORTANT BIT LIKE 1037.84137 AND
FLIP THE LAST BIT 7 TO 8. OR I MAKE AN ERROR AND CREATE A NAN

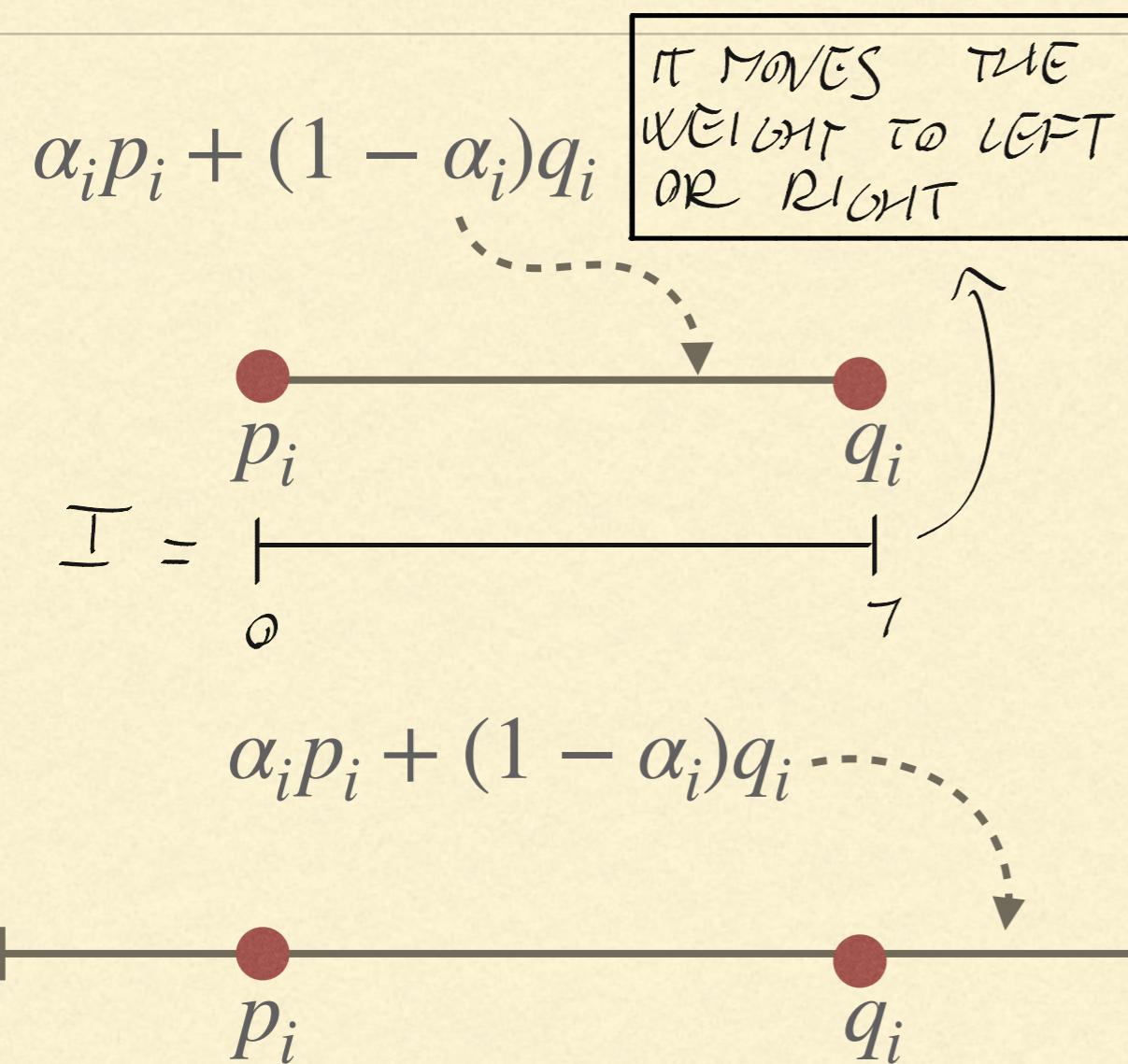
- If each gene is a floating point value then mutation and crossover should be adapted

WHAT WE NEED? CHANGE THE FITNESS VALUE *

SELECTION (PHENOTYPE) DON'T CARE, BUT MUTATION DOES (GENOTYPES)

Crossover

$$\begin{aligned}
 p_1 &= 3 \\
 q_1 &= 5 \\
 \alpha_1 &= 0.5
 \end{aligned}
 \quad
 \begin{aligned}
 &>= 0.5(3) + (1-0.5)5 \\
 &= 1.5 + 2.5 = 4
 \end{aligned}$$



Intermediate recombination

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

JUST WEIGHT THE VALUES WITH SOME PROBABILITY

$$\begin{aligned}
 p_1 &= 3 \\
 q_1 &= 5 \\
 \alpha_1 &= 2
 \end{aligned}
 \quad
 \begin{aligned}
 &>= 2(3) + (1-2)5 \\
 &= 6 + 5 = 11
 \end{aligned}$$

Line recombination

$\alpha_i \leftarrow \text{random}(-k, 1 + k)$
I ALLOW THE NUMBER TO MOVE OUTSIDE THE TWO INDIVIDUALS

MUTATION

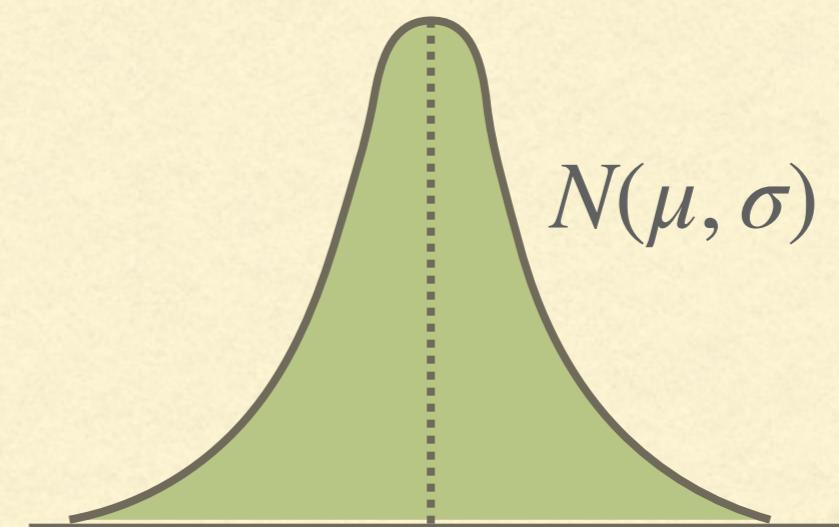
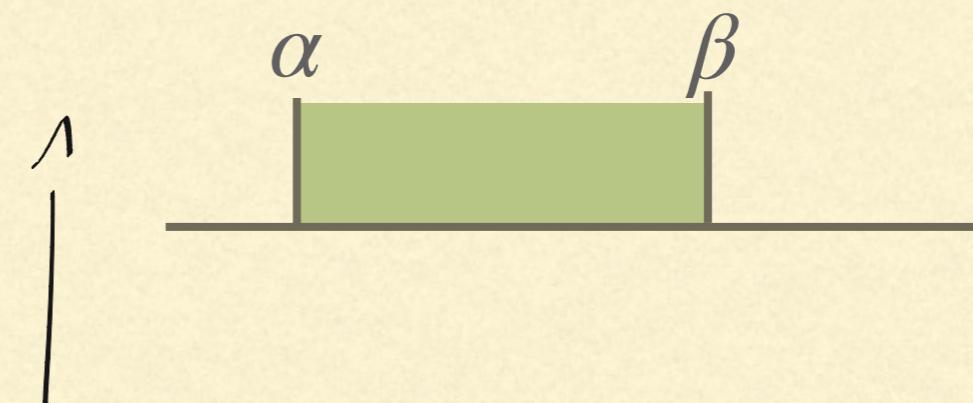
INSTEAD OF BIT FLIP YOU
ADD A (SMALL) PERTURBATION

Add a small value
to each coordinate
of the individual



$p + \epsilon$ → GIVEN BY OR →

Uniform between two values



POTENTIALLY YOU CAN MIX
CONTINUOUS AND DISCRETE.
IT DOESN'T AFFECT SELECTION,
BUT MUTATION NEED A PERTURBATION
ADAPTED TO IF IS DISCRETE AN
CONTINUOUS.

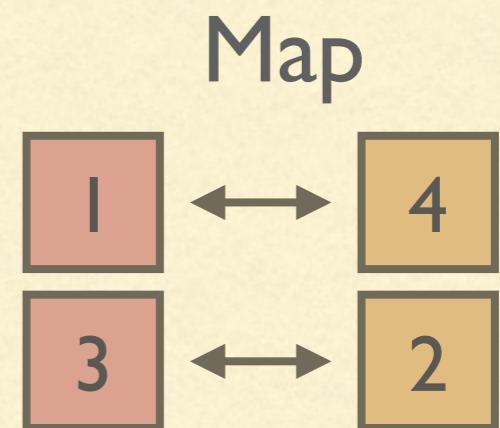
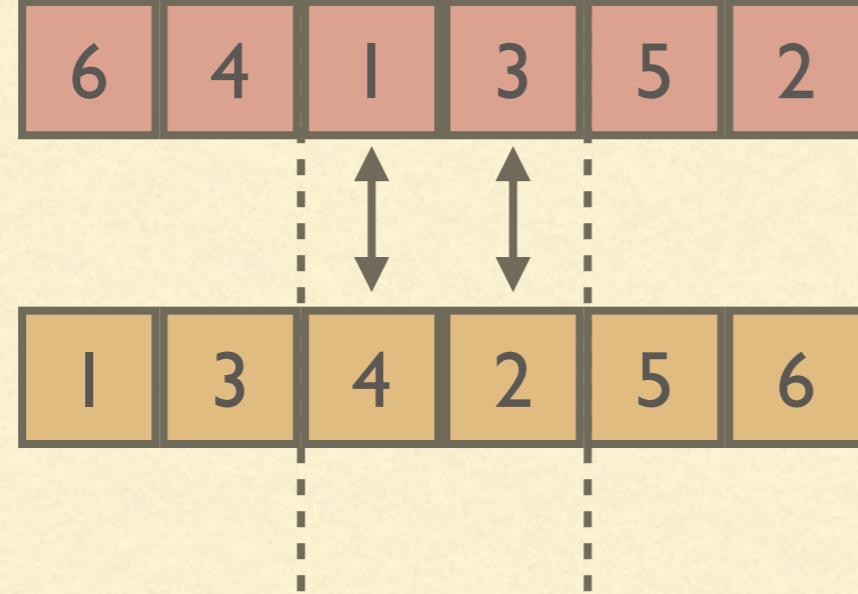
CYCLE AND PMX CROSSOVERS

- 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

HOW DO YOU MAINTAIN? REPEAT CROSSOVER UNTIL IT RESPECT THE CONSTRAINTS
BUT IT'S INEFFICIENT OR ASSIGN AN EXTREMELY BAD FITNESS THAT NOT
FIT CONSTRAINTS BUT IT'S STILL INEFFICIENT

PARTIALLY MAPPED CROSSOVER (PMX)

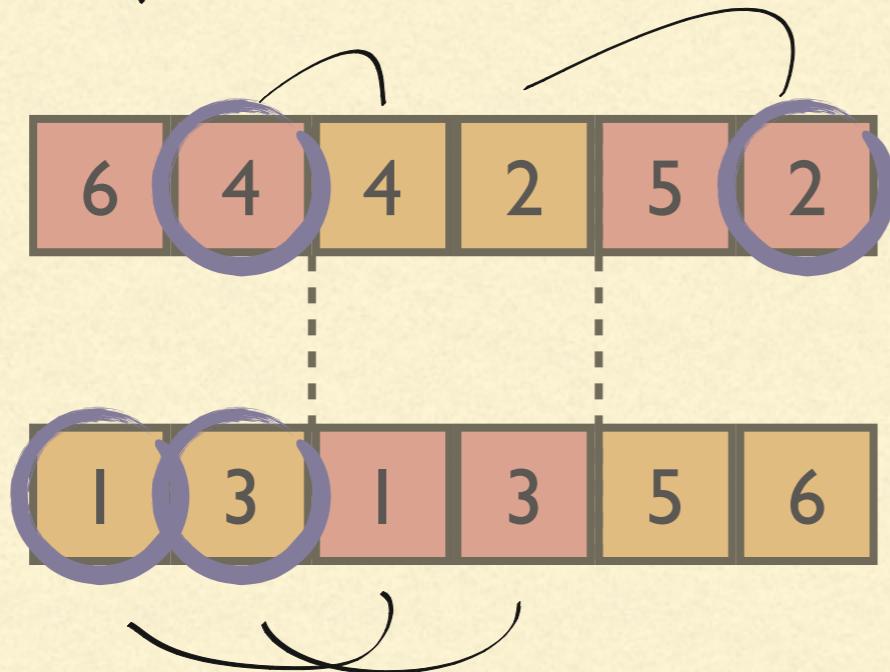
A 2 POINTS CROSSOVER
AND THEN A CORRECTION PROCEDURE



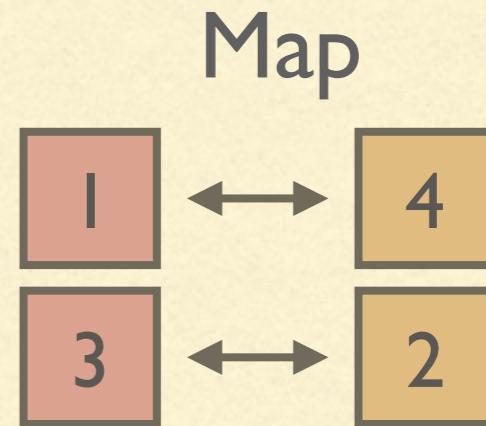
We select two crossover point and we build a map

PARTIALLY MAPPED CROSSOVER (PMX)

WE HAVE TO CHANGE THE NUMBER THAT REPEAT 2 TIMES

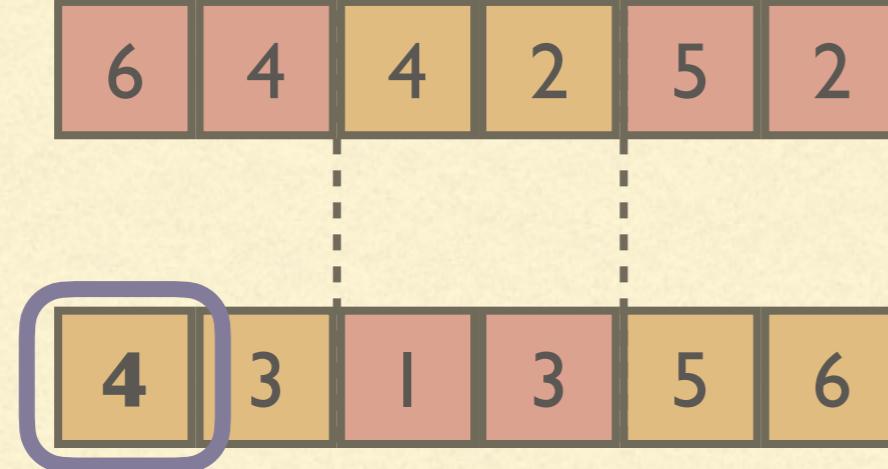


WE NEED
TO SWAP
THEM. HOW?

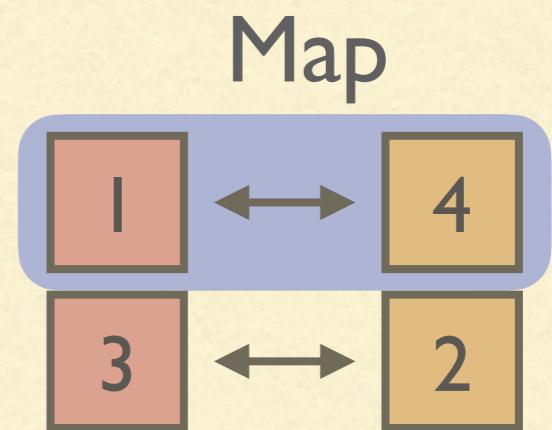


We perform the exchange but the offspring are not valid

PARTIALLY MAPPED CROSSOVER (PMX)

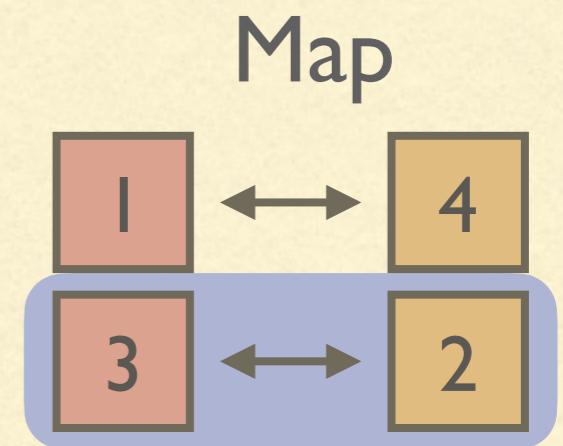
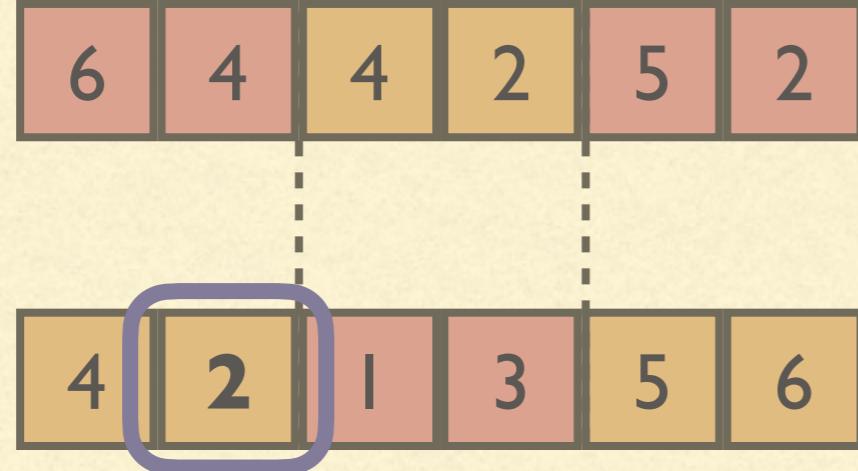


WE JUST
USE THE
VALUE



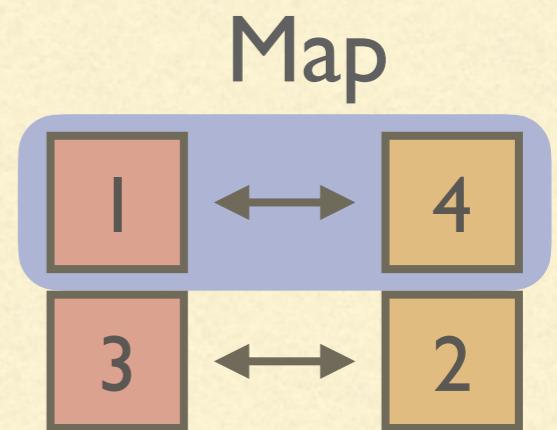
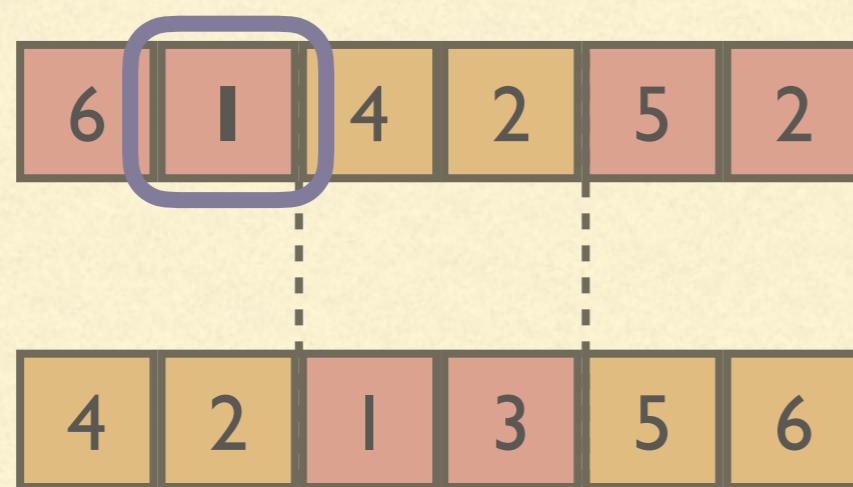
We iterate the map until we have resolved the conflicts

PARTIALLY MAPPED CROSSOVER (PMX)



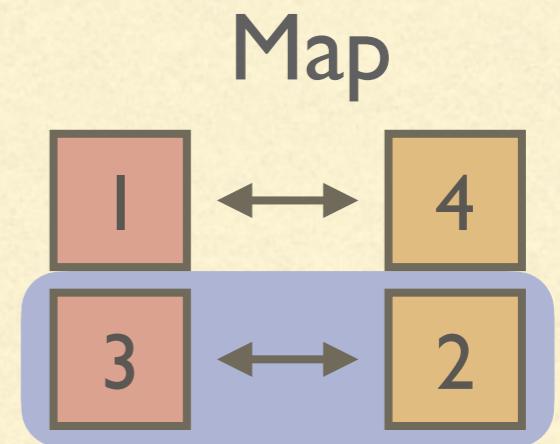
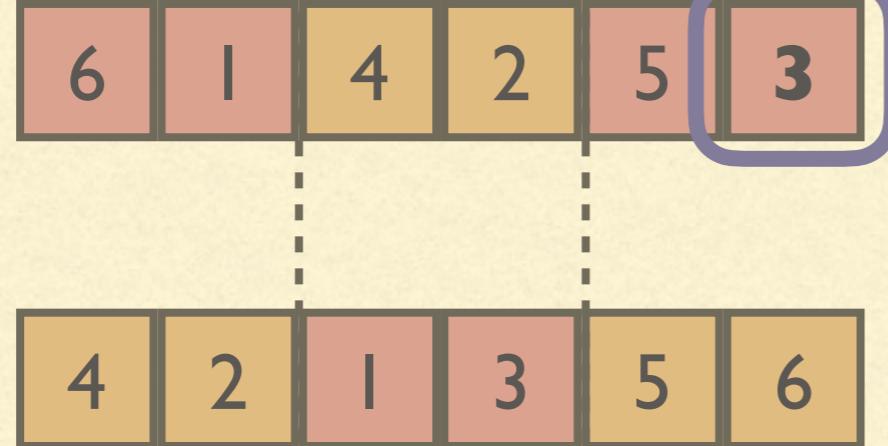
We iterate the map until we have resolved the conflicts

PARTIALLY MAPPED CROSSOVER (PMX)



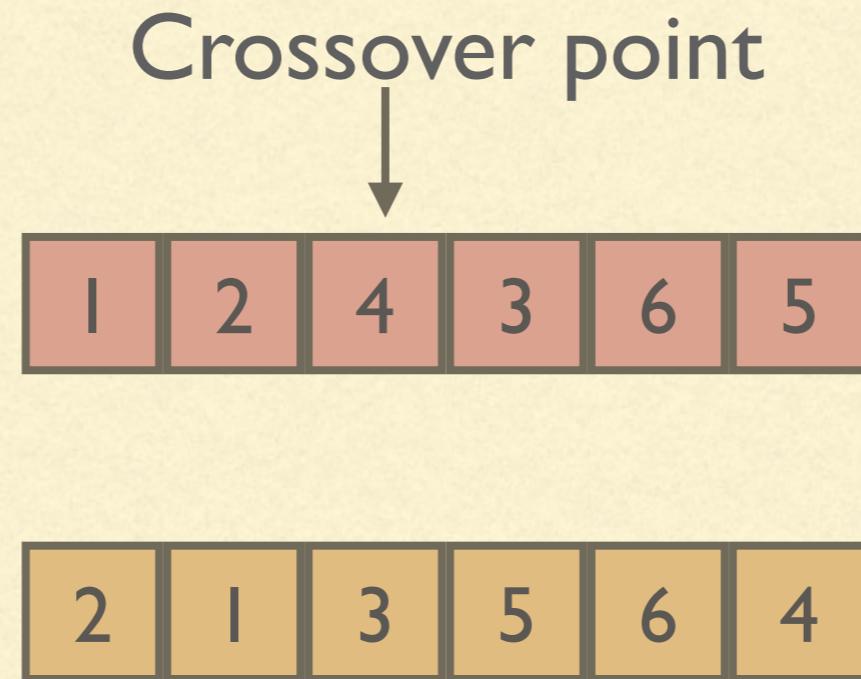
We iterate the map until we have resolved the conflicts

PARTIALLY MAPPED CROSSOVER (PMX)



We iterate the map until we have resolved the conflicts

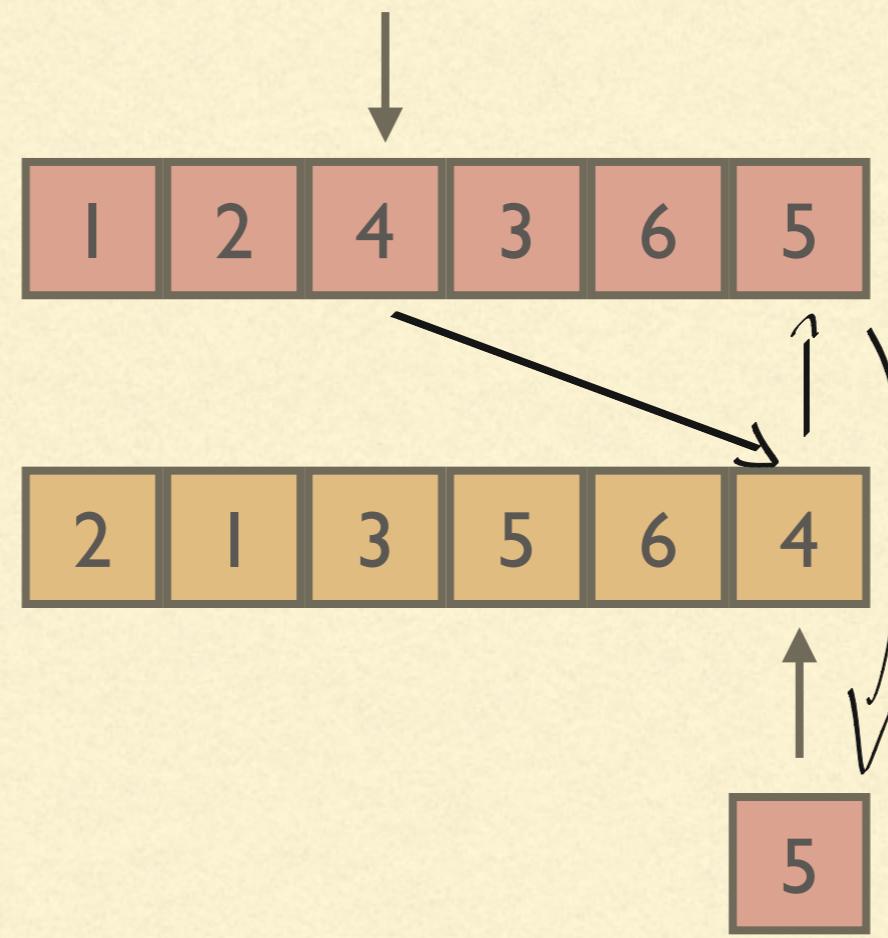
CYCLE CROSSOVER



SO, WE HAVE 2 TYPE OF PERMUTATION:
THE FIRST BETWEEN INDIVIDUALS
AND THE SECOND BETWEEN NUMBERS INSIDE
THE SINGLE INDIVIDUAL

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

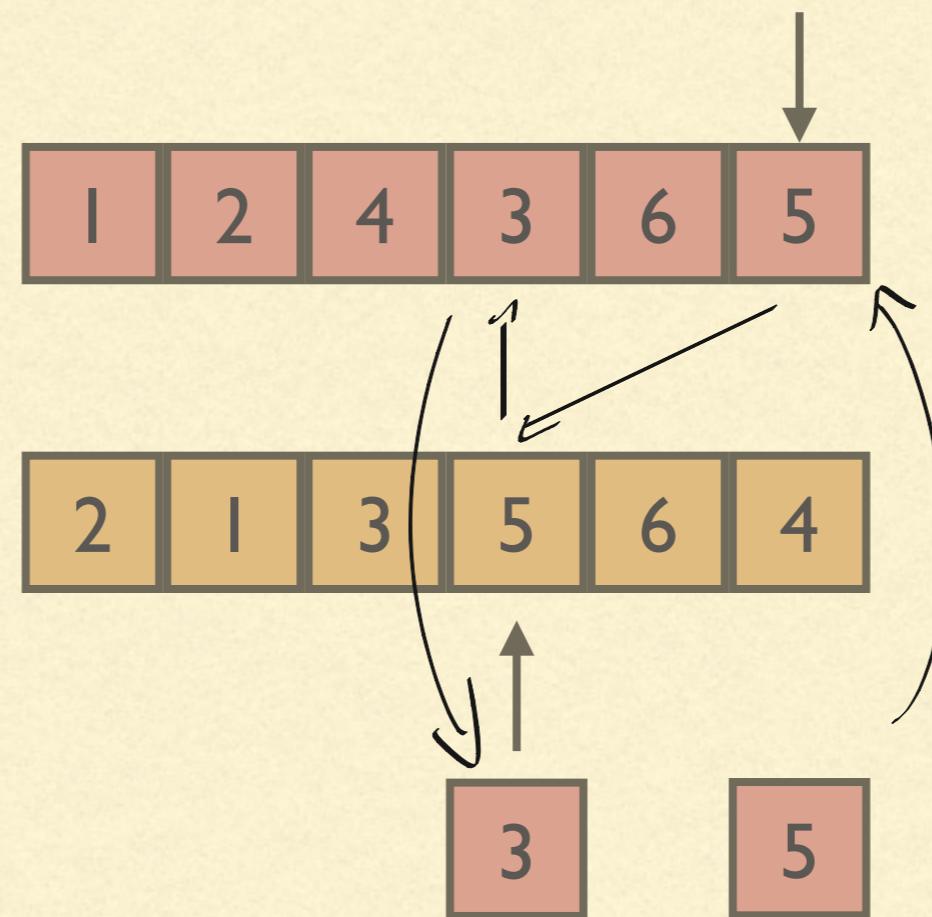
CYCLE CROSSOVER



- VALUE FIRST SAMPLE
- SEARCH THE THAT VALUE ON THE SECOND SAMPLE
- EXCHANGE THE VALUE THAT HAVE THE SAME POSITION IN THE FIRST SAMPLE WITH THE POSITION OF VALUE WE FIND ABOVE
- USE THE VALUE WE EXTRACT AS INDEX
- REPEAT

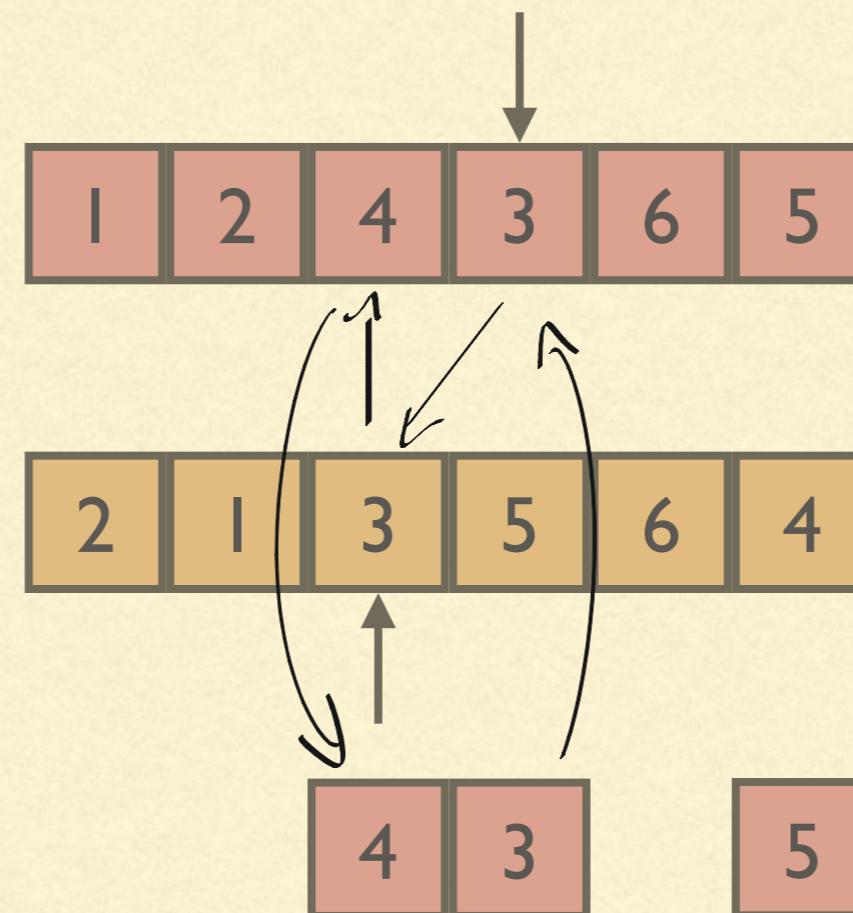
Once found we copy the value from the first parent.
Repeat until we return to the beginning

CYCLE CROSSOVER



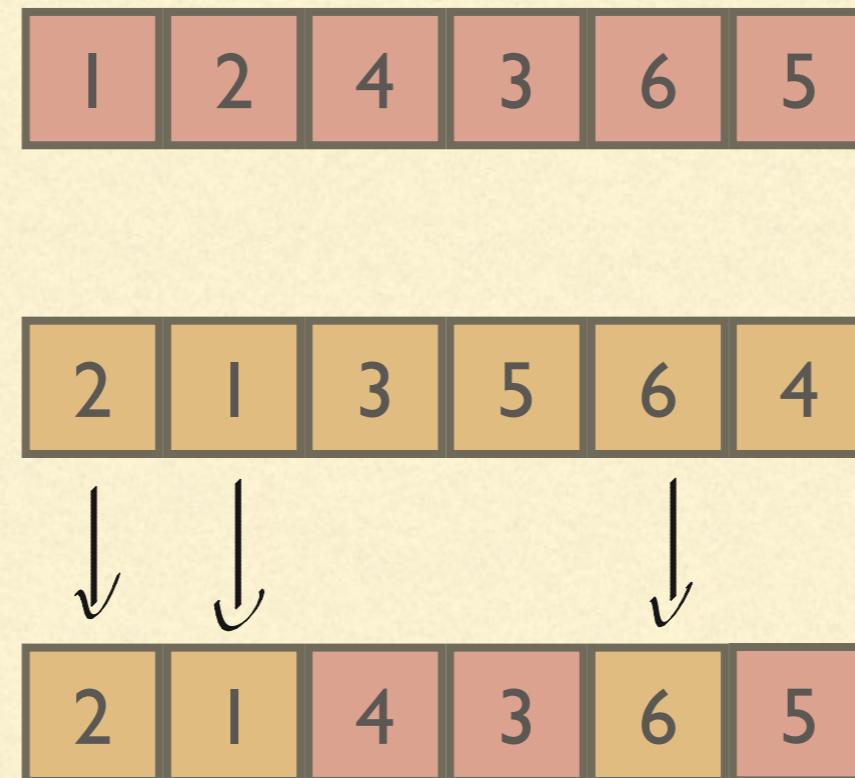
Once found we copy the value from the first parent.
Repeat until we return to the beginning

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

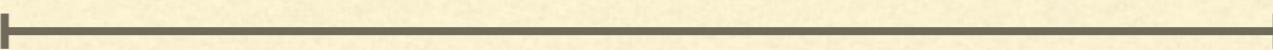
REPRESENTING GRAPHS

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

IT CAN BE USEFUL BECAUSE THE “DEVICE” CAN BE EXTREMELY SIMPLE THEN THE GRAPH AND IT CAN BE EASIER TO REPRESENT REGULARITIES
EX: BUILT A GRAPH AND REPEAT IT 20 TIMES

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

$$V = \{a, b, c, d, e\}$$

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

Possible mutations:

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

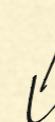
BUT HOW WE PERFORM AN
INTERACTIONS BETWEEN SOLUTION?

Each one can have
a different probability

NON EXISTING EDGE



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



IF USE A LINEAR REPRESENTATION
ROW BY ROW OR PERFORM A 2D
CROSSOVER ON THE MATRIX

SELECTION PHASE : GOOD SOLUTION PROPAGATE AND BAD FADE AWAY

PRODUCTION RULES

THIS ONE IS AN INDIRECT REPRESENTATION

- 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

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

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

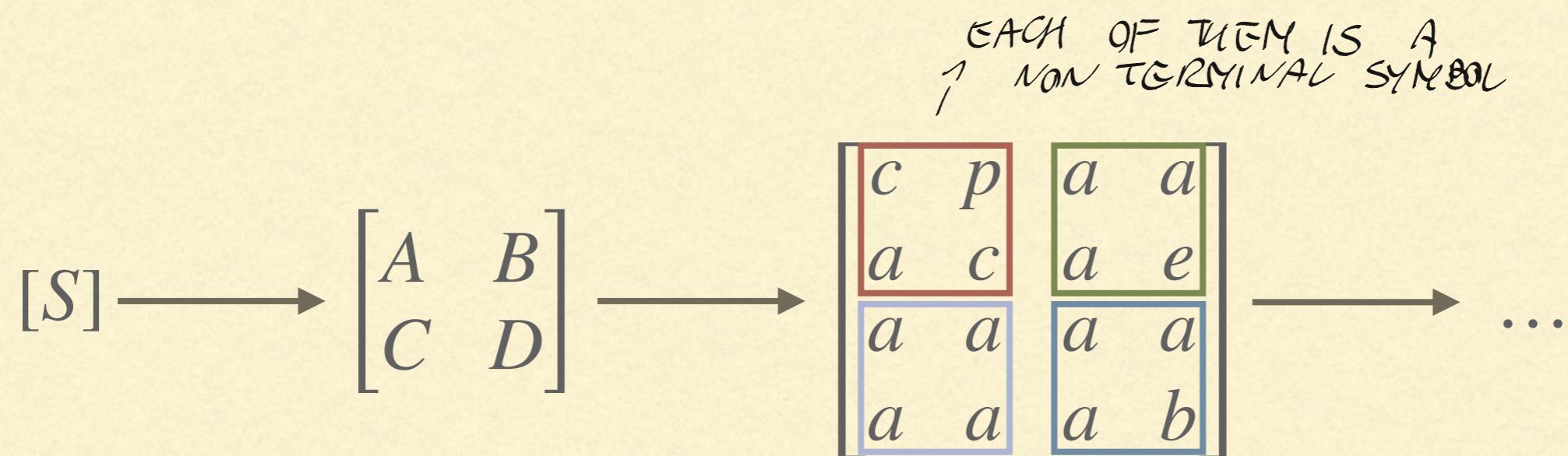
$$a \rightarrow \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad b \rightarrow \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \quad c \rightarrow \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \quad e \rightarrow \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \quad p \rightarrow \begin{bmatrix} 1 & 1 \\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

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



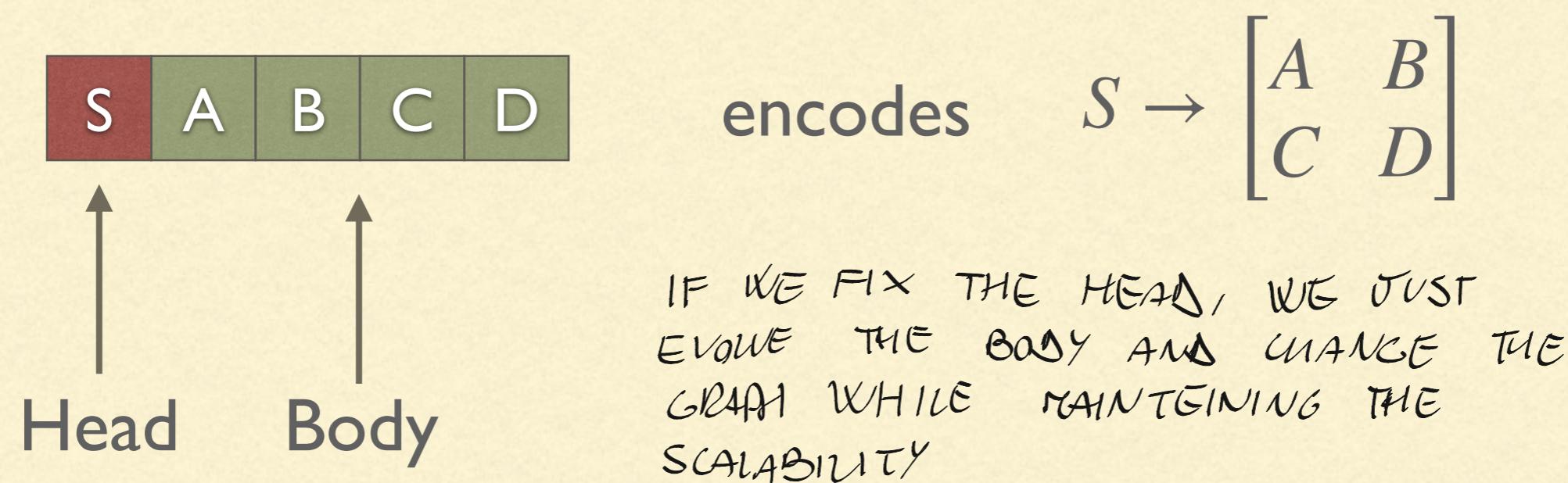
PRODUCTION RULES

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)

IF I CAN GROW A SET OF PRODUCTION
RULES, THEN I CAN GENERATE VERY LARGE
GRAPH

PRODUCTION RULE: ENCODING



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