BABEŞ-BOLYAI UNIVERSITY
Faculty of Computer Science and Mathematics

ARTIFICIAL INTELLIGENCE

Solving search problems

Informed local search strategies

Evolutionary Algorithms

Topics

A. Short introduction in Artificial Intelligence (AI)

A. **Solving search problems**
   A. **Definition of search problems**
   B. **Search strategies**
      A. Uninformed search strategies
      B. Informed search strategies
      C. **Local search strategies (**Hill Climbing, Simulated Annealing, Tabu Search**, Evolutionary algorithms**, PSO, ACO)
      D. Adversarial search strategies

C. Intelligent systems
   A. Rule-based systems in certain environments
   B. Rule-based systems in uncertain environments (Bayes, Fuzzy)
   C. Learning systems
      A. Decision Trees
      B. Artificial Neural Networks
      C. Support Vector Machines
      - Evolutionary algorithms
   D. Hybrid systems

Content

- Solving problems by search
  - Informed search strategies
    - Local strategies
      - Evolutionary algorithms

Useful information

- Chapter 14 of *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- *M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, 1998*
- Chapter 7.6 of *A. A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- Chapter 9 of *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

Local search

- Typology
  - Simple local search – a single neighbour state is retained
    - Hill climbing → selects the best neighbour
    - Simulated annealing → selects probabilistic the best neighbour
    - Tabu search→ retains the list of visited solutions
  - Beam local search – more states are retained (a population of states)
    - Evolutionary algorithms
    - Particle swarm optimisation
    - Ant colony optmisation

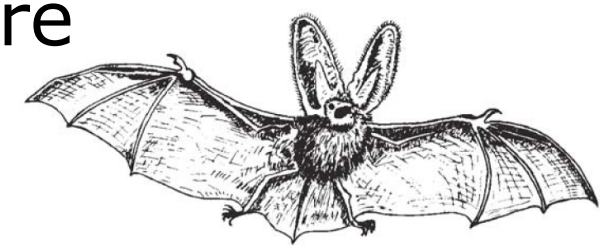
Nature-inspired search

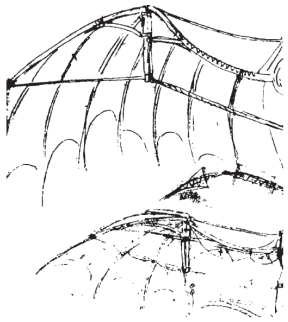
- Best method for solving a problem
  - Human brain
    - Has created the wheel, car, town, etc.
  - Mechanism of evolution
    - Has created the human brain

- Simulation of nature
  - By machines' help → the artificial neural networks simulate the brain
    - Flying vehicles, DNA computers, membrane-based computers
  - By algorithms' help
    - Evolutionary algorithms simulate the evolution of nature
    - Particle Swarm Optimisation simulates the collective and social behaviour
    - Ant Colony Optimisation

Evolutionary Algorithms (EAs) – Basic elements

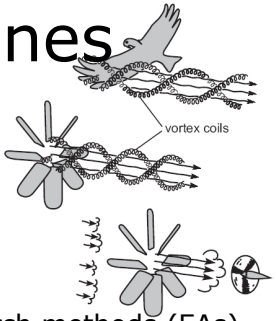
- Simulation of nature
  - Fly of bats





  - Leonardo da Vinci – sketch of a flying machine

  - Flies of birds and planes
  - Flies of birds and wind-turbines



EAs – basic elements

- Main characteristics of EAs
  - Iterative and parallel processes
  - Based on random search
  - Bio-inspired – involve mechanisms as:
    - Natural selection
    - Reproduction
    - Recombination
    - Mutation

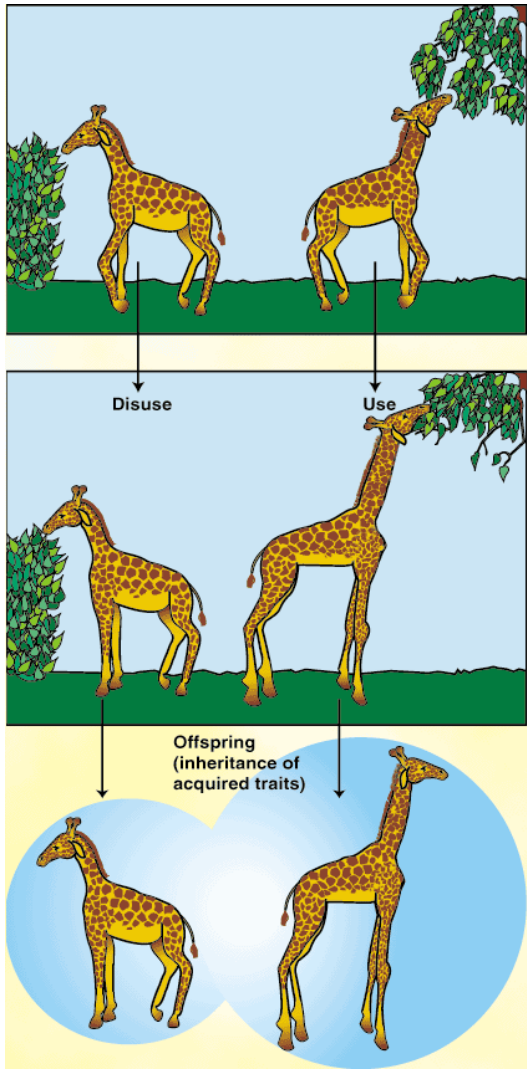
EAs – basic elements

Historical points

- Jean Baptise de Lamark (1744-1829)
  - Has proposed in 1809 an explanation

  For origin of species in the book

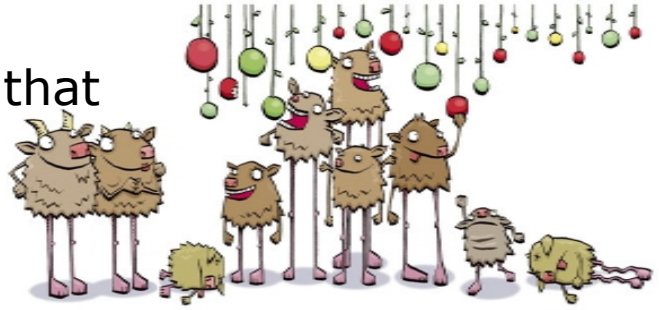
  Zoological Philosophy:
    - Needs of an organism determine the evolving characteristics
    - Useful characteristics could be transferred to offspring
  - *use and disuse law*

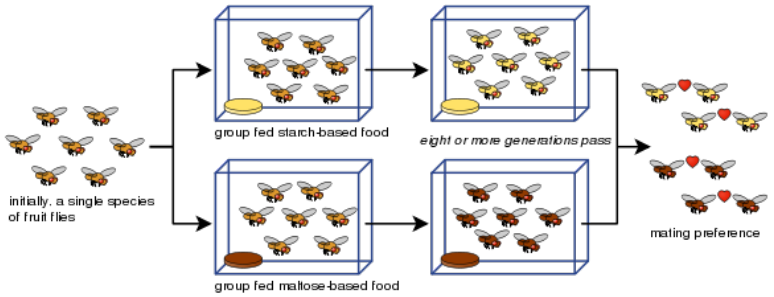


EAs – basic elements

Historical points

- Charles Darwin (1807-1882)
  - In the book *Origin of Species* he proved that all the organisms have evolved based on:
    - Variation
      - Overproduction of offspring
    - Natural selection
      - Competition (generation of constant size)
      - Fitness survival
      - Reproduction
      - Occurrence of new species





EAs – basic elements

Historical points

- Modern theory of evolution
  - Darwin's theory is improved by mechanism of genetic inheritance
  - Genetic variance is produced by
    - Mutation and
    - Sexual recombination
  - L. Fogel 1962 (San Diego, CA)– *Evolutionary Programming (EP)*
  - J. Holland 1962 (Ann Arbor, MI) → *Genetic Algorithms* (GAs)
  - I. Rechenberg & H.-P. Schwefel 1965 (Berlin, Germany)→ *Evolution Strategies (ESs)*
  - J. Koza 1989 (Palo Alto, CA) → *Genetic Programming (GP*)

EAs – basic elements

- Evolutionary metaphor

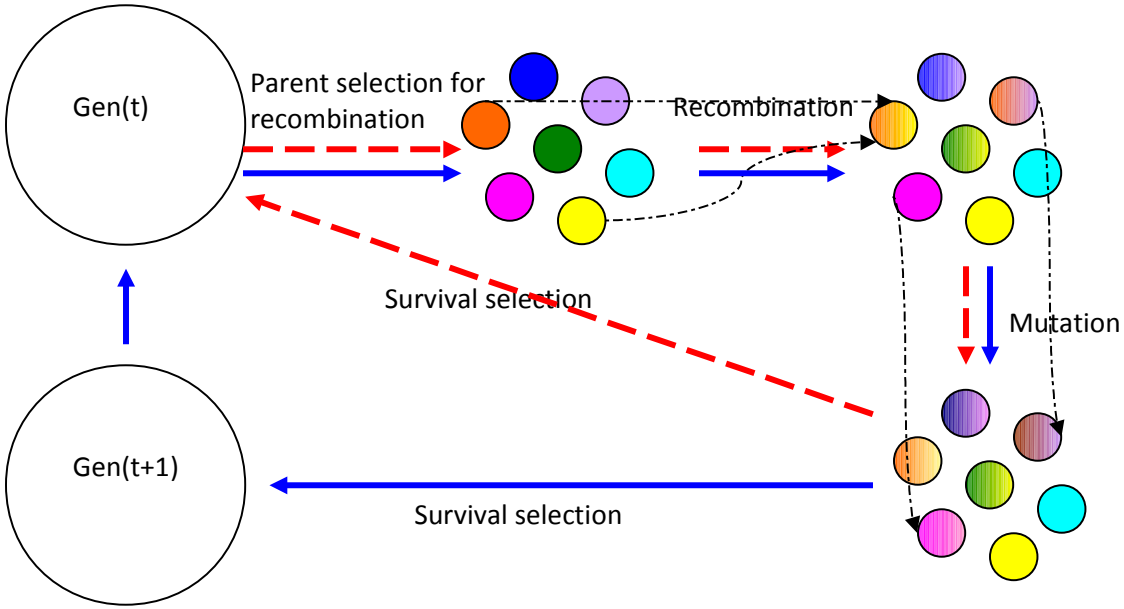
Natural evolution		Problem solving
Individual	↔	Possible solution
Population	↔	Set of possible solutions
Chromosome	↔	Coding of a possible solution
Gene	↔	Part of coding
Fitness	↔	Quality
Crossover and Mutation	↔	Search operators
Environment	↔	Problem

EAs - algorithm

- General sketch
- Design

EAs - algorithm

- General sketch of an EA
  - Generational
  - Steady-state



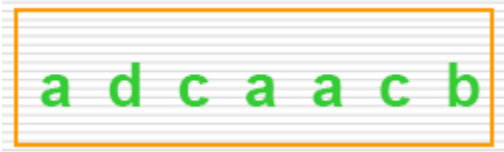
EAs - algorithm

- Design
  - Chromosome representation
  - Population model
  - Fitness function
  - Genetic operators
    - Selection
    - Mutation
    - Crossover
  - Stop condition

EAs - algorithm
Design – representation

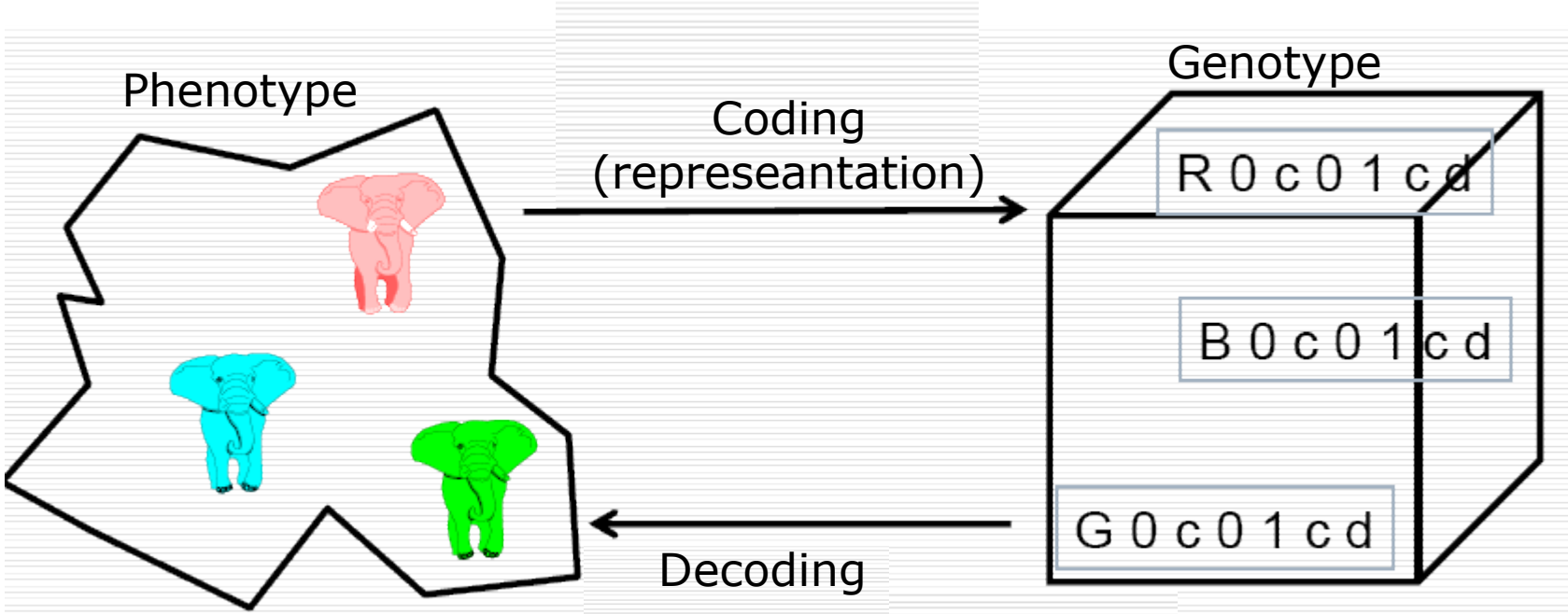
- 2 levels of each possible solution
  - External level → phenotype
    - Individual – original object in the context of the problem
    - The possible solutions are evaluated here
    - Ant, knapsack, elephant, towns, ...

  - Internal level → genotype
    - Chromosome – code associated to an object
      - Composed by genes, located in loci (fix positions) and having some values (alleles)
    - The possible solutions are searched here
    - One-dimensional vector (with numbers, bits, characters), matrix, …



EAs - algorithm design – representation

- Representation must be representative for:
  - Problem
  - Fitness function and
  - Genetic operators



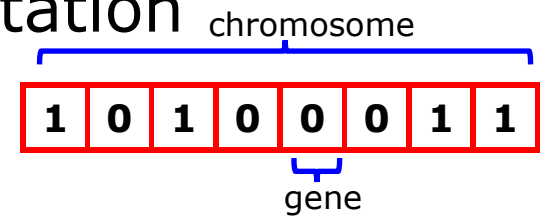
EAs - algorithm
Design - representation

Typology of chromosome's representation

- Linear
  - Discrete
    - Binary → knapsack problem
    - Not-binary
      - Integers
        - Random → image processing
        - Permutation → travelling salesman problem (TSP)
      - Class-based → map colouring problem
  - Continuous (real) → function optimization
- Tree-based → regression problems

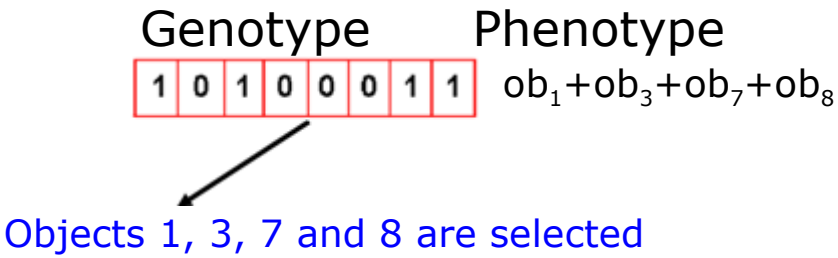
EAs - algorithm
Design - representation

- Linear discrete and binary representation
  - Genotype
    - Bit-strings



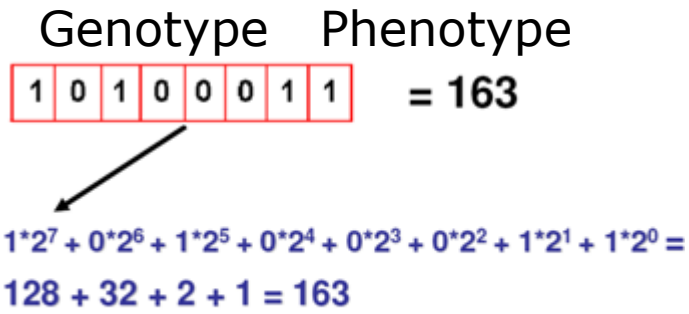
EAs - algorithm
Design - representation

- Linear discrete and binary representation
  - Genotype
    - Bit-strings
  - Phenotype
    - Boolean elements
      - Eg. Knapsack problem – selected objects for the bag



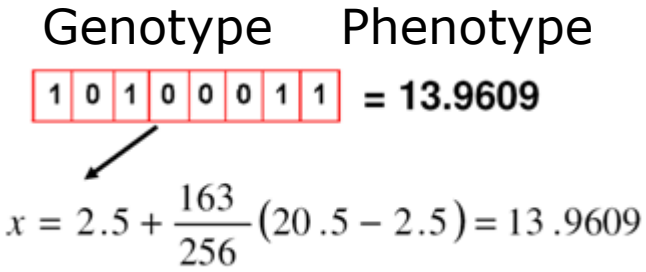
EAs - algorithm
Design - representation

- Linear discrete and binary representation
  - Genotype
    - Bit-strings
  - Phenotype
    - Boolean elements
      - Eg. Knapsack problem – selected objects for the bag
    - Integers



EAs - algorithm
Design - representation

- Linear discrete and binary representation
  - Genotype
    - Bit-strings
  - Phenotype
    - Boolean elements
      - Example: Knapsack problem – selected objects for the bag
    - Integers
    - Real numbers from a range (ex. [2.5, 20.5])



EAs - algorithm
Design - representation

Transformation of real values from binary representation

- Let be $z \in [x,y] \subseteq \mathcal{R}$ represented as $\{a_1,\ldots,a_L\} \in \{0,1\}^L$
- Function $[x,y] \rightarrow \{0,1\}^L$ must be inverse (a phenotype corresponds to a genotype)
- Function Γ: $\{0,1\}^L \rightarrow [x,y]$ defines the representation

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

- Remarks
  - 2^L values can be represented
  - L – maximum precision of solution
  - For a better precision → long chromosomes → slowly evolution

EAs - algorithm
Design - representation

- Linear discrete non-binary integer random representation
  - Genotype
    - Vector of integers from a given range
  - Phenotype
    - Utility of numbers in the problem
  - Example: Pay a sum S by using different n coins
    - Genotype → vector of n integers from range [0, S/value of current coin]
    - Phenotype → how many coins of each type must be considered

EAs - algorithm
Design - representation

- Linear discrete non-binary integer permutation representation
  - Genotype
    - Permutation of n elements (n – number of genes)
  - Phenotype
    - Utility of permutation in problem
  - Example Traveling Salesman Problem
    - Genotype → permutation of n elements
    - Phenotype → visiting order of towns (each town has associated a number from {1,2,...,*n*})

EAs - algorithm
Design - representation

- Linear discrete non-binary integer class-based representation
  - Similarly to integer one, but labels are used instead numbers
  - Genotype
    - Vector of labels from a given set
  - Phenotype
    - Labels' meaning
  - Example Map colouring problem
    - Genotype → vector of n colours (n – number of countries)
    - Phenotype → what colour has to be used for each country

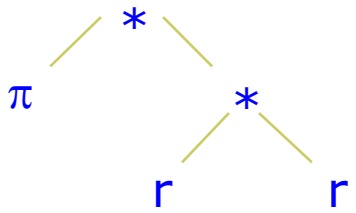
EAs - algorithm
Design - representation

- Linear continuous (real) representation
  - Genotype
    - Vector of real numbers
  - Phenotype
    - Number meaning
  - Example Function optimisation $f: R^n \rightarrow R$
    - Genotype → more real numbers $X=[x_1, x_2, \ldots, x_n]$, $x_i \in R$
    - Phenotype → values of function f arguments

EAs - algorithm
Design - representation

- Tree-based representation
  - Genotype
    - Trees than encode S-expressions
    - Internal nodes →functions (F)
      - Mathematical
        - Arithmetic operators
        - Boolean operators
      - Statements
        - Of a given programming language
        - Of other language type
    - Leaf → terminals (T)
      - Real or Boolean values, constants or variables
      - Sub-programs
  - Phenotype
    - Meaning of S-expressions
  - Example Computing the circle area

$$\pi * r^2$$



EAs - algorithm
Design – creation of population

- Population – concept
  - Aim
    - To keep a collection of possible solutions (candidate solutions)
      - Repetitions are allowed
  - Properties
    - (usually) fixed dimension μ
    - Diversity
      - Number of different fitnesses/phenotypes/genotypes
  - Remarks
    - Represents the basic unit that evolves
      - The entire population evolves, not only the individuals!!!

EAs - algorithm
Design – creation of population

- Population - initialisation
  - Uniformly distributed in the search space (if it is possible)
    - Binary strings
      - Randomly generation of 0 and 1 with a 0.5 probability (fifty-fifth)
    - Arrays of real numbers uniformly generated (in a given range)
    - Permutations
      - Generation of identical permutation and making some changes

EAs - algorithm
Design – creation of population

- Population - initialisation
  - Uniformly distributed in the search space (if it is possible)
    - Trees
      - *Full* method – complete trees
        - Nodes of depth d < D_{max} are randomly initialised by a function from function set F
        - nodes of depth d = D_{max} are randomly initialised by a terminal from the terminal set T
      - *Grow* method – incomplete trees
        - Nodes of depth d < D_{max} are randomly initialised by an element from *F U T*
        - nodes of depth d = D_{max} are randomly initialised by a terminal from the terminal set T
      - *Ramped half and half* method
        - ½ of population is initialised by *Full* methods
        - ½ of population is initialised by *Grow* methods
        - By using different depths

EAs - algorithm
Design – creation of population

- Population model:
  - Generational EA
    - Each generation creates μ offspring
    - Each individual survives a generation only
    - Set of parents is totally replaced by set of offspring
  - Steady-state EA
    - Each generation creates a single offspring
    - A single parent (the worst one) is replaced by the offspring

- Generation Gap
  - Proportion of replaced population
  - $1 = \mu/\mu$, for generational model
  - $1/\mu$, for steady-state model

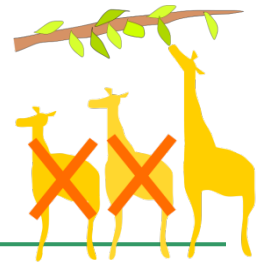
EAs - algorithm
Design – fitness function

- Aim
  - Reflects the adaptation to environment
  - Quality function or objective function
  - Associates a value to each candidate solution
    - Consequences over selection → the more different values, the better
- Properties
  - Costly stage
    - Un - changed individuals could not be re-evaluated
- Typology:
  - Number of objectives
    - One-objective
    - Multi-objective → Pareto fronts
  - Optimisation direction
    - Maximisation
    - Minimisation
  - Degree of precision
    - Deterministic
    - Heuristic

EAs - algorithm
Design – fitness function

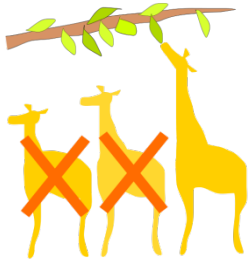
- Examples
  - Knapsack problem
    - Representation → linear, discrete and binary
    - Fitness → abs(knapsack's capacity – weight of selected objects) → min
  - Problem of paying sum s by using different coins
    - Representation → linear, discrete and integer
    - Fitness → abs(sum to be paid – sum of selected coins) → min
  - TSP
    - Representation → linear, discrete, integer, permutation
    - Fitness → cost of path → min
  - Numerical function optimization
    - Representation → linear, continuous, real
    - Fitness → value of function → min/max
  - Computing the circle's area
    - Representation → tree-based
    - Fitness → sum of square errors (difference between the real value and the computed value for a given set of examples) → min

EAs - algorithm
Design – selection



- Aim:
  - Gives more reproduction/survival chances to better individuals
    - Weaker individuals have chances also because they could contain useful genetic material
  - Orients the population to improve its quality
- Properties
  - Works at population-level
  - Is based on fitness only (is independent to representation)
  - Helps to escape from local optima (because its stochastic nature)

EAs - algorithm
Design – selection

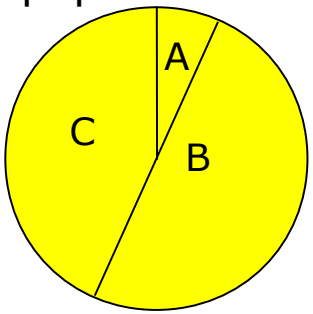


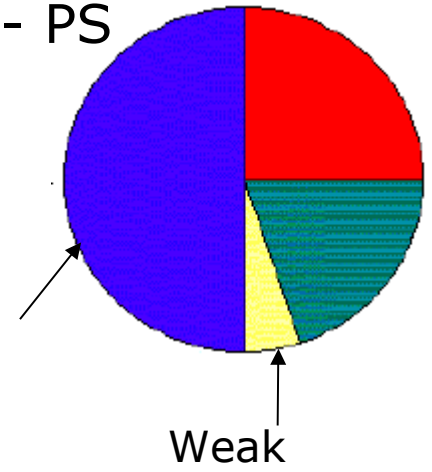
- Typology
  - Aim
    - Parent selection (from current generation) for reproduction
    - Survival selection (from parents and offspring) for next generation
  - Winner strategy
    - Deterministic – the best wins
    - Stochastic – the best has more chances to win
  - Mechanism
    - Selection for recombination
      - Proportional selection (based on fitness) } Based on entire population
      - Rank-based selection
      - Tournament selection ----→ Based on a part of population
    - Survival selection
      - Age-based selection
      - Fitness-based selection

EAs - algorithm
Design – recombination selection

- Proportional selection (fitness-based selection) - PS
  - Main idea
    - Roulette algorithm for entire population
    - Estimation of the copies # of an individual (selection pressure)

      $E(n_i) = \mu \frac{f(i)}{\langle f \rangle}$, where:
    - μ = size of population,
    - *f(i)* = fitness of individual *i*,
    - $\langle f \rangle$ = mean fitness of population
  - Better individuals
    - Have more space on roulette
    - Have more chances to be selected
  - Ex. A population of μ = 3 individuals



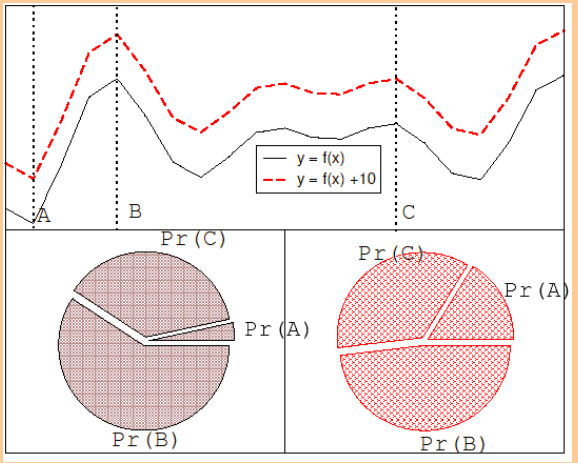


	f(i)	$P_{selPS}(i)$
A	1	1/10=0.1
B	5	5/10=0.5
C	4	4/10=0.4
Suma	10	1

EAs - algorithm
Design – recombination selection

Proportional selection (fitness-based selection) – PS

- Advantages
- Simple algorithm
- Disadvantages
- Premature convergence
  - Best chromosomes predispose to dominate the population
- Low selection pressure when fitness functions are very similar (at the end of a run)
- Real results are different to theoretical probabilistic distribution
- Works at the entire population level
- Solutions
- Fitness scaling
  - Windowing
    - $f'(i) = f(i) - \beta^t$, where β is a parameter that depends on evolution history
      - eg. β is the fitness of the weakest individual of current population (the t^{th} generation)
  - Sigma scaling (Goldberg type)
    - $f'(i) = \max\{f(i) - (\langle f \rangle - c * \sigma_f), 0.0\}$, where:
      - C – a constant (usually, 2)
      - $\langle f \rangle$ - average fitness of population
      - σ_f – standard deviation of population fitness
  - Normalisation
    - Starts by absolute (initial) fitnesses
    - Standardize these fitnesses such as the fitnesses:
      - Belong to [0,1]
      - Best fitness is the smallest one (equal to 0)
      - Sum of them is 1
- Another selection mechanism



EAs - algorithm
Design – selection for recombination

- Ranking selection – RS
  - Main idea
    - Sort the entire population based on fitness
      - Increases the algorithm complexity, but it is negligible related to the fitness evaluation
    - Each individual receives a rank
    - Computes the selection probabilities based on these ranks
      - Best individual has rank μ
      - Worst individual has rank 1
    - Tries to solve the problems of proportional selection by using relative fitness (instead of absolute fitness)

EAs - algorithm
Design – selection for recombination

- Ranking selection - RS
  - Ranking procedures
    - Linear (LR) $P_{lin\_rank}(i) = \frac{2-s}{\mu} + \frac{2i(s-1)}{\mu(\mu-1)}$
      - s – selection pressure
        - Measures the advantages of the best individual
        - $1.0 < s \leq 2.0$
        - In the generational algorithm s represents the copies number of an individual
      - Eg. For a population of μ = 3 individuals

	f(i)	$P_{selPS}(i)$	Rank	$P_{selLR}(i)$ for s=2	$P_{selRL}(i)$ for s=1.5
A	1	1/10=0.1	1	0.33	0.33
B	5	5/10=0.5	3	1.00	0.33
C	4	4/10=0.4	2	0.67	0.33
Sum	10	1			

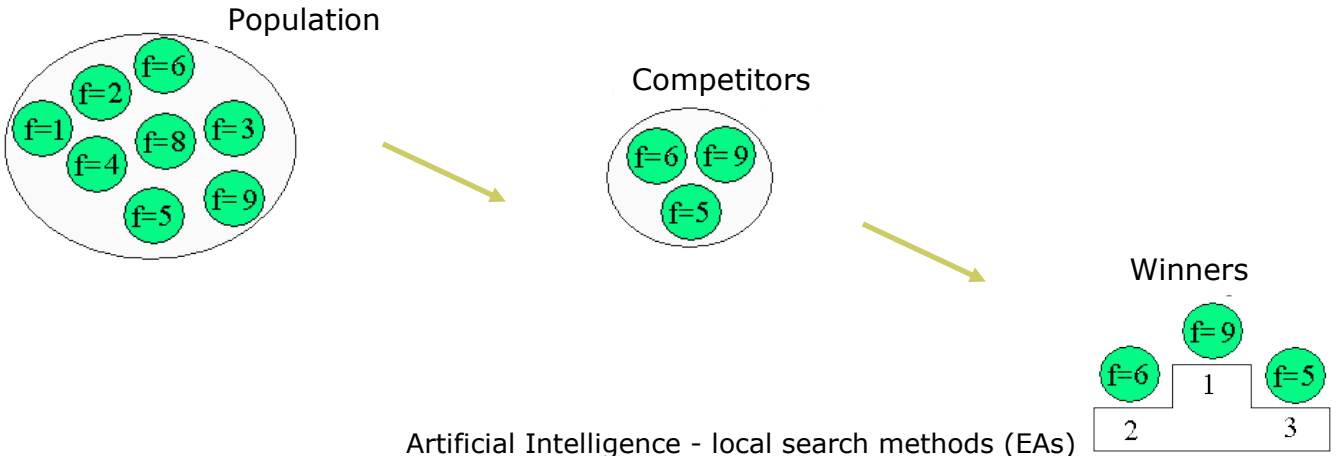
    - Exponential (ER) $P_{\exp\_rank}(i) = \frac{1-e^{-i}}{c}$
      - Best individual can have more than 2 copies
      - C – normalisation factor
        - Depends on the population size (μ)
        - Must be choose such as the sum of selection probabilities to be 1

EAs - algorithm
Design – selection for recombination

- Ranking selection - RS
  - Advantages
    - Keep the selection pressure constant
  - Disadvantages
    - Works with the entire population
  - Solutions
    - Another selection procedure

EAs - algorithm
Design – selection for recombination

- Tournament selection
  - Main idea
    - Chooses k individuals → sample of k individuals (k – tournament size)
    - Selects the best individual of the sample
    - Probability of sample selection depends on
      - Rank of individual
      - Sample size (k)
        - The larger k is, the greater selection pressure is
      - Choosing manner – with replacement (steady-state model) or without replacement
        - Selection without replacement increases the selection pressure
      - For k =2 the time required by the best individual to dominate the population is the same to that from linear ranking selection with s = 2 * *p*, *p* – selection probability of the best individual from population



EAs - algorithm
Design – selection for recombination

- Tournament selection
  - Advantages
    - Does not work with the entire population
    - Easy to implement
    - Easy to control the selection pressure by using parameter k
  - Disadvantages
    - The real results of this selection are different to theoretical distribution (similarly to roulette selection)

EAs - algorithm
Design – survival selection

- Survival selection (selection for replacement)
  - Based on age
    - Eliminates the oldest individuals
  - Based on fitness
    - Proportional selection
    - Ranking selection
    - Tournament selection
    - Elitism
      - Keep the best individuals from a generation to the next one (if the offspring are weaker than parents, then keep the parents)
    - GENITOR (replaces the worst individual)
      - Elimination of the worst λ individuals

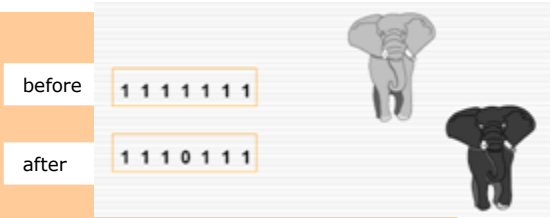
EAs - algorithm
Design – variation operators

- Aim :
  - Generation of new possible solutions
- Properties
  - Works at individual level
  - Is based on individual representation (fitness independent)
  - Helps the exploration and exploitation of the search space
  - Must produce valid individuals
- Typology
  - Arity criterion
    - Arity 1 → mutation operators
    - Arity > 1 → recombination/crossover operators

EAs - algorithm
Design – mutation

- Aim
  - Reintroduces in population the lost genetic material
  - Unary search operator (continuous space)
  - Introduces the diversity in population (discrete space)

- Properties
  - Works at genotype level
  - Based on random elements



  - Responsible to the exploration of promising regions of the search space
  - Responsible to escape from local optima

  - Must introduce small and stochastic changes for an individual
  - Size of mutation must be controllable
  - Can probabilistic take place (by a given probability p_m) at the gene level

EAs - algorithm Design – mutation

- Typology
  - Binary representation
    - Strong mutation – bit-flipping
    - Weak mutation
  - Integer representation
    - Random resetting
    - Creep mutation
  - Permutation representation
    - Insertion mutation
    - Swap mutation
    - Inverse mutation
    - scramble mutation
    - K-opt mutation
  - Real representation
    - Uniform mutation
    - Non-uniform mutation
      - Gaussian mutation
      - Cauchy mutation
      - Laplace mutation
  - Tree-based representation → future lecture
    - Grow mutation
    - Shrink mutation
    - Switch mutation
    - Cycle mutation
    - Koza mutation
    - Mutation for numerical terminals

EAs - algorithm
Design – mutation (binary representation)

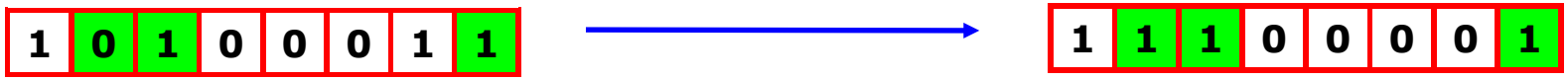
- A chromosome $c=(g_1,g_2,...,g_L)$ becomes $c'=(g_1',g_2',...,g_L')$, where $g_i, g_i' \in \{0,1\}$, for $i=1,2,...,L$

- Strong mutation – *bit flipping*
  - Main idea
    - Changes by probability p_m (mutation rate) all the genes in their complement
      - 1 → 0
      - 0 → 1
    - Eg. A chromosome of $L = 8$ genes, $p_m = 0.1$

EAs - algorithm
Design – mutation (binary representation)

- A chromosome $c=(g_1,g_2,...,g_L)$ becomes $c'=(g_1',g_2',...,g_L')$, where $g_i, g_i' \in \{0,1\}$, for $i=1,2,...,L$

- Weak mutation
  - Main idea
    - Changes by probability p_m (mutation rate) some of the genes in 0 or 1
      - 1 → 0/1
      - 0 → 1/0
    - Eg. A chromosome of $L = 8$ genes, $p_m = 0.1$



EAs - algorithm
Design – mutation (integer representation)

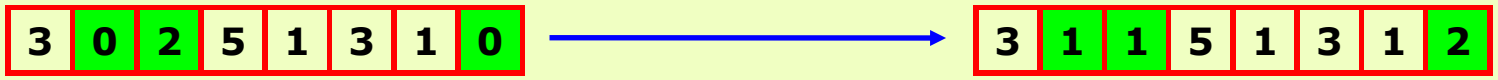
- A chromosome $c=(g_1,g_2,...,g_L)$ becomes $c'=(g_1',g_2',...,g_L')$, where $g_i, g_i' \in \{val_1, val_2,...,val_k\}$ for $i=1,2,...,L$

- *Random resetting* mutation
  - Main idea
    - The value of a gene is changed (by probability p_m) into another value (from the definition domain)

EAs - algorithm
Design – mutation (integer representation)

- A chromosome $c=(g_1,g_2,...,g_L)$ becomes $c'=(g_1',g_2',...,g_L')$, where $g_i, g_i' \in \{val_1, val_2, ..., val_k\}$, for i=1,2,...,L

- *Creep* mutation
  - Main idea
    - The value of a gene is changed (by probability p_m) by adding a positive/negative value
      - New value follows a 0 symmetric distribution
      - The performed change is very small



EAs - algorithm
Design – mutation (permutation representation)

- A chromosome $c=(g_1,g_2,...,g_L)$ with $g_i \neq g_i$ for all $i \neq i$ becomes $c'=(g_1',g_2',...,g_L')$, where $g_i, g_i' \in \{val_1, val_2,...,val_L\}$, for $i=1,2,...,L$ s.a. $g_i' \neq g_i'$ for all $i \neq i$.

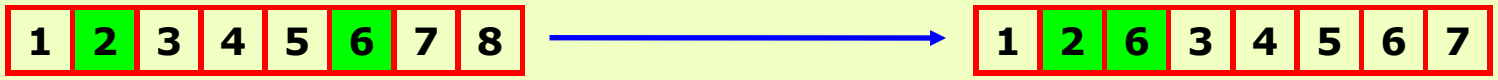
- *Swap mutation*
  - Main idea
    - Randomly choose 2 genes and swap their values

EAs - algorithm
Design – mutation (permutation representation)

- A chromosome $c=(g_1,g_2,...,g_L)$ with $g_i \neq g_i$ for all $i \neq i$ becomes $c'=(g_1',g_2',...,g_L')$, where $g_i, g_i' \in \{val_1, val_2,...,val_L\}$, for $i=1,2,...,L$ s.a. $g_i' \neq g_i'$ for all $i \neq i$.

Insertion mutation

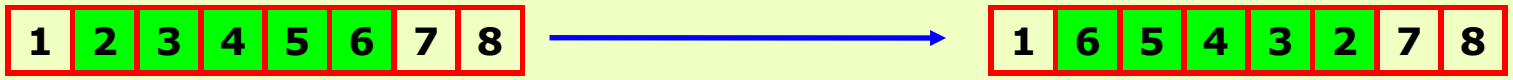
- Main idea
  - Randomly choose 2 genes g_i and g_j with $j > i$
  - Insert gene gj after gene gi s.a. $g_i'=g_i$, $g_{i+1}'=g_j$, $g_{k+2}'=g_{k+1}$, for $k=i, i+1, i+2, ...$



EAs - algorithm
Design – mutation (permutation representation)

- A chromosome $c=(g_1,g_2,...,g_L)$ with $g_i \neq g_i$ for all $i \neq i$ becomes $c'=(g_1',g_2',...,g_L')$, where $g_i, g_i' \in \{val_1, val_2,...,val_L\}$, for $i=1,2,...,L$ s.a. $g_i' \neq g_i'$ for all $i \neq i$.

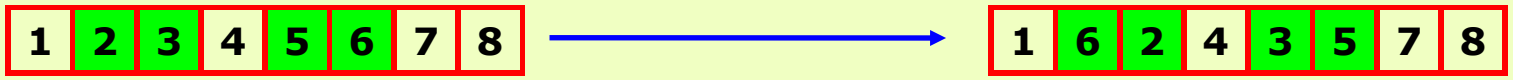
- Inversion mutation
  - Main idea
    - Randomly choose 2 genes and inverse the order of genes between them (sub-string of genes)



EAs - algorithm
Design – mutation (permutation representation)

- A chromosome $c=(g_1,g_2,...,g_L)$ with $g_i \neq g_i$ for all $i \neq i$ becomes $c'=(g_1',g_2',...,g_L')$, where $g_i, g_i' \in \{val_1, val_2,...,val_L\}$, for $i=1,2,...,L$ s.a. $g_i' \neq g_i'$ for all $i \neq i$.

- *scramble mutation*
  - Main idea
    - Randomly choose a (continuous or discontinuous) sub-array of genes and re-organise that genes



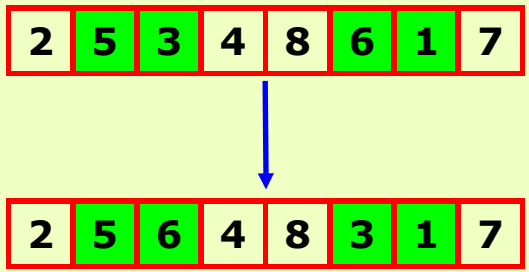
EAs - algorithm
Design – mutation (permutation representation)

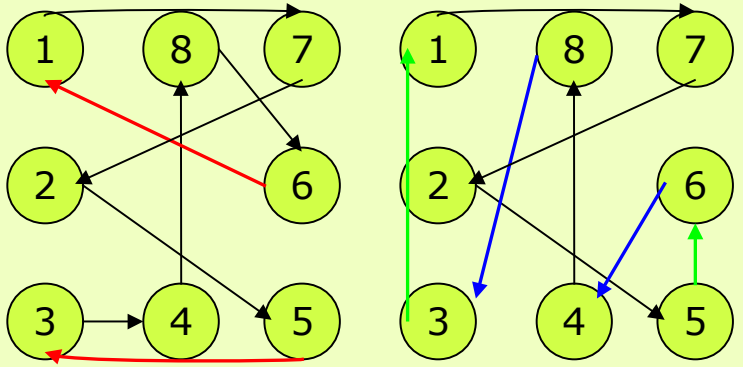
- A chromosome $c=(g_1,g_2,...,g_L)$ with $g_i \neq g_i$ for all $i \neq i$ becomes $c'=(g_1',g_2',...,g_L')$, where $g_i, g_i' \in \{val_1, val_2,...,val_L\}$, for $i=1,2,...,L$ s.a. $g_i' \neq g_i'$ for all $i \neq i$.

K-opt mutation

- Main idea
  - Choose 2 disjoint sub-strings of length k
  - Interchange 2 elements of these sub-strings

k=2





EAs - algorithm
Design – mutation (real representation)

- A chromosome $c=(g_1,g_2,...,g_L)$ becomes $c'=(g_1',g_2',...,g_L')$, where $g_i, g_i' \in [LI_i, LS_i]$, for $i=1,2,...,L$

- Uniform mutation
  - Main idea
    - g_i' is changed by probability p_m into a new value that is randomly uniform generated in $[LI_i, LS_i]$ range

EAs - algorithm
Design – mutation (real representation)

- A chromosome $c=(g_1,g_2,...,g_L)$ becomes $c'=(g_1',g_2',...,g_L')$, where $g_i, g_i' \in [LI_i, LS_i]$, for $i=1,2,...,L$

- Non-uniform mutation
  - Main idea

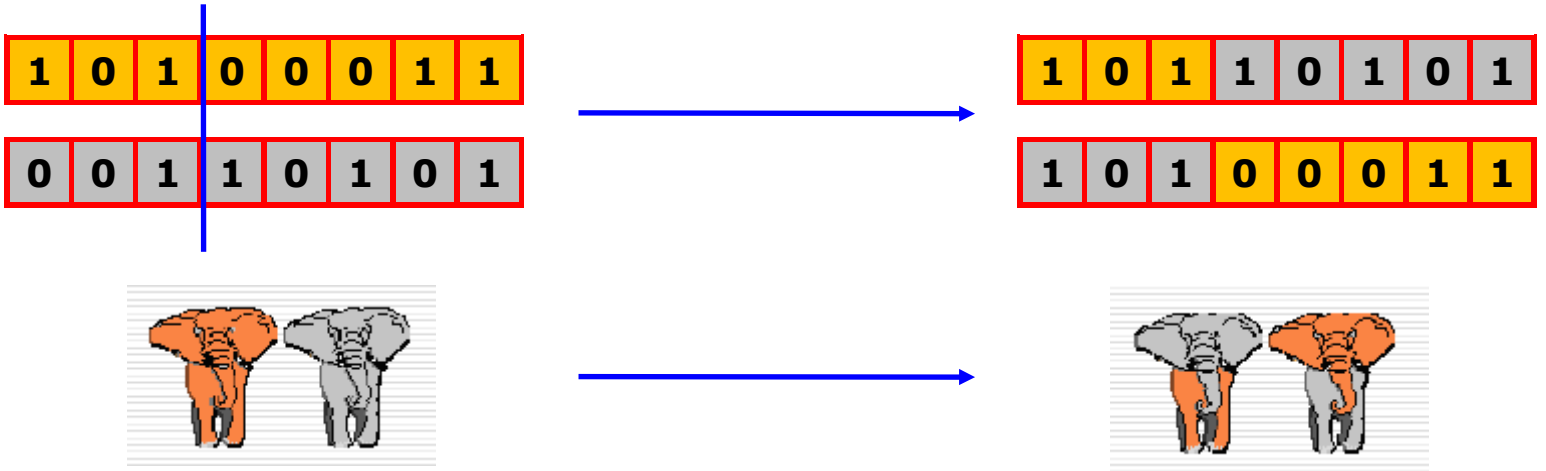
The value of a gene is changed by adding a positive/negative value with a given probability (p_m)

- The added value belongs to a distribution of type
  - $N(\mu, \sigma)$ (Gaussian) with $\mu = 0$
  - Cauchy (x_0, γ)
  - Laplace (μ, b)
- And it is re-introduced in $[LI_i, LS_i]$ range (if it is necessary) – *clamping*

EAs - algorithm
Design – recombination

- Aim
  - Mix the parents' information

- Properties
  - The offspring has to inherit something from both parents
    - Selection of mixed information is randomly performed
  - Operator for exploitation of already discover possible solutions
  - The offspring can be better, the same or weaker than their parents
  - Its effects are reducing while the search converges



EAs - algorithm Design – recombination

- Typology
  - Binary and integer representation
    - With cutting points
    - Uniforme
  - Permutation representation
    - Order crossover (version 1 and version 2)
    - Partially Mapped Crossover
    - Cycle crossover
    - Edge-based crossover
  - Real representation
    - Discrete
    - Arithmetic
      - Singular
      - Simple
      - Complete
    - Geometric
    - Shuffle crossover
    - Simulated binary crossover
  - Tree-based representation
    - Sub-tree based crossover → future lecture

EAs - algorithm
Design – recombination (binary and integer representation)

- From 2 parent chromosomes
  - $p_1=(g_1^1,g_2^1,...,g_L^1)$ and $p_2=(g_1^2,g_2^2,...,g_L^2)$
- 2 offspring are obtained
  - $c_1 =(g_1',g_2',...,g_L')$ and $c_2 =(g_1'',g_2'',...,g_L'')$,
  - where $g_i^1,g_i^2, g_i', g_i'' \in \{0,1\} / \{val_1, val_2, ..., val_k\}$, for i=1,2,...,L

N-cutting point crossover

- Main idea
  - Choose n cutting-points ($n < L$)
  - Cut the parents through these points
  - Put together the resulted parts, by alternating the parents

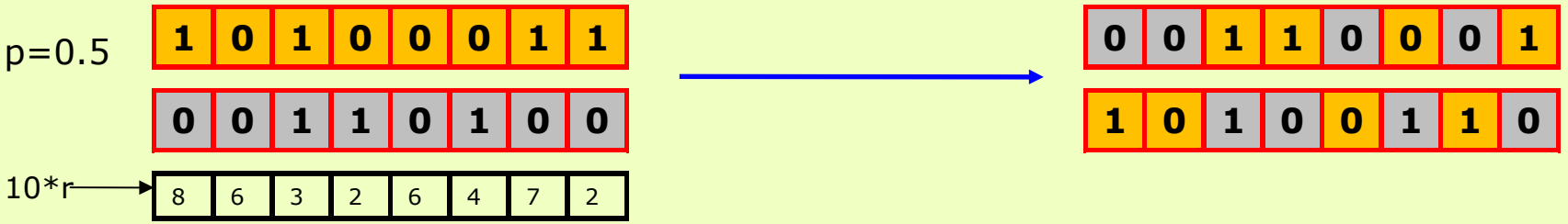
EAs - algorithm
Design – recombination (binary and integer representation)

- N cutting point crossover
  - Properties
    - Average of values encoded by parents = average of values encoded by offspring
      - Eg binary representation on 4 bits of integer numbers – XO with n=1 after second bit
        - p_1 = (1,0,1,0), p_2 = (1,1,0,1)
        - c_1 = (1,0, 0,1), c_2 = (1,1,1,0)
        - val(p_1) = 10, val(p_2) = (13) ➔ (val(p_1) + val(p_2))/2 = 23/2=11.5
        - val(c_1) = 9, val(c_2) = (14) ➔ (val(c_1) + val(c_2))/2 = 23/2=11.5
      - Eg. Binary representation on 4 bits for knapsack problem (K=10, 4 items of weight and value: (2,7), (1,8), (3,1), (2,3))
        - p_1 = (1,0,1,0), p_2 = (1,1,0,1)
        - c_1 = (1,0, 0,1), c_2 = (1,1,1,0)
        - val(p_1) = 8, val(p_2) = 18 ➔ (val(p_1) + val(p_2))/2 = 26/2=13
        - val(c_1) = 10, val(c_2) = 16 ➔ (val(c_1) + val(c_2))/2 = 26/2=13
    - Probability of $\beta \approx 1$ is the largest one $\beta = \left| \frac{val(d_1) - val(d_2)}{val(p_1) - val(p_2)} \right|$
      - Contracting crossover $\beta < 1$
        - Offspring values are between parent values
      - Expanding crossover $\beta > 1$
        - Parent values are between offspring values
      - Stationary crossover $\beta = 1$
        - Offspring values are equal to parent values

EAs - algorithm
Design – recombination (binary and integer representation)

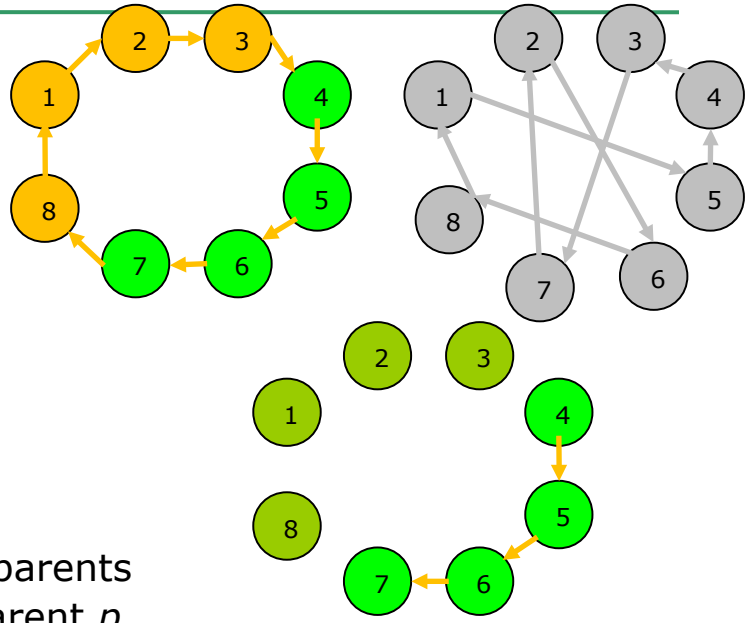
- From 2 parent chromosomes
  - $p_1=(g_1^1,g_2^1,...,g_L^1)$ and $p_2=(g_1^2,g_2^2,...,g_L^2)$
- 2 offspring are obtained
  - $c_1 =(g_1',g_2',...,g_L')$ and $c_2 =(g_1'',g_2'',...,g_L'')$,
  - where g_i^1,g_i^2, g_i', $g_i'' \in \{0,1\} / \{val_1, val_2, ..., val_k\}$, for i=1,2,...,L

- Uniform crossover
  - Main idea
    - Each gene of an offspring comes from a randomly and uniform selected parent:
      - For each gene a uniform random number *r* is generated
      - If *r* < probability *p* (usually, *p*=0.5), c_1 will inherit that gene from p_1 and c_2 from p_2,
      - otherwise, c_1 will inherit p_2 and c_2 will inherit p_1

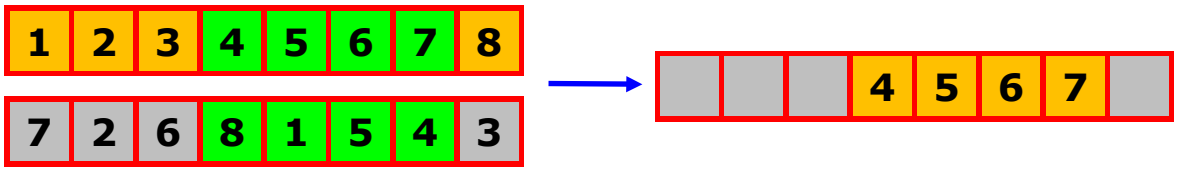


EAs - algorithm
Design – recombination (permutation representation)

- From 2 parent chromosomes
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ and $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- 2 offspring are obtained
  - $c_1 =(g_1',g_2',\ldots,g_L')$ and $c_2 =(g_1'',g_2'',\ldots,g_L'')$,
  - Where $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, for i=1,2,...,L

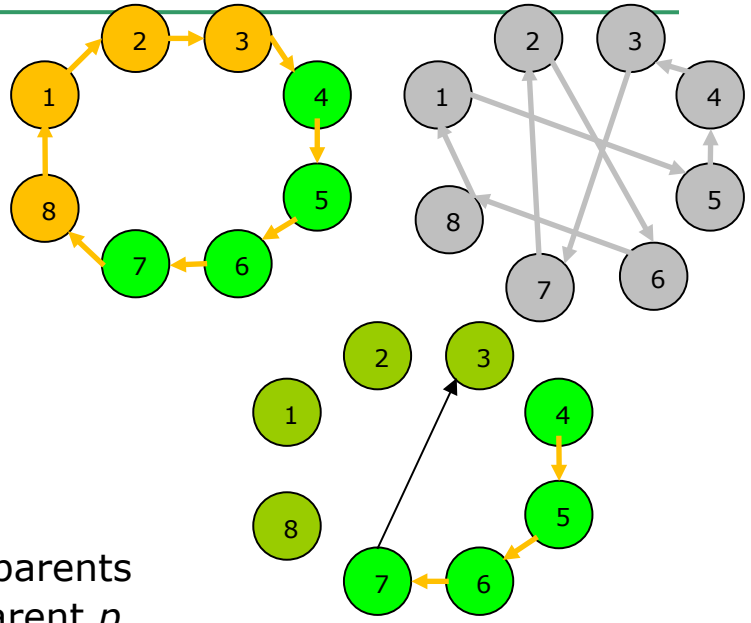


- Order crossover
  - Main idea
    - Offspring keep the order of genes from parents
    - Choose a substring of genes from the parent p_1
    - Copy the substring from p_1 into offspring d_1 (on corresponding positions)

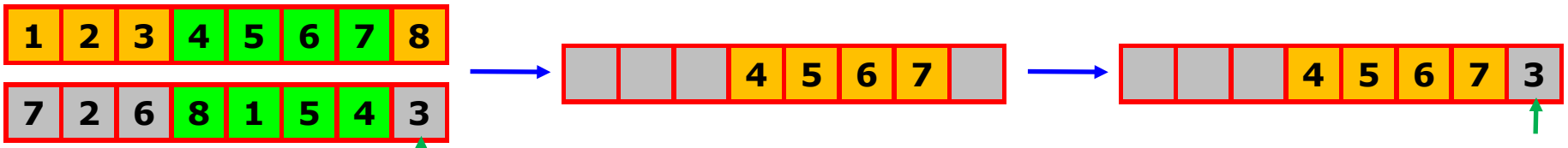


EAs - algorithm
Design – recombination (permutation representation)

- From 2 parent chromosomes
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ and $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- 2 offspring are obtained
  - $c_1=(g_1',g_2',\ldots,g_L')$ and $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - Where $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, for i=1,2,…,L

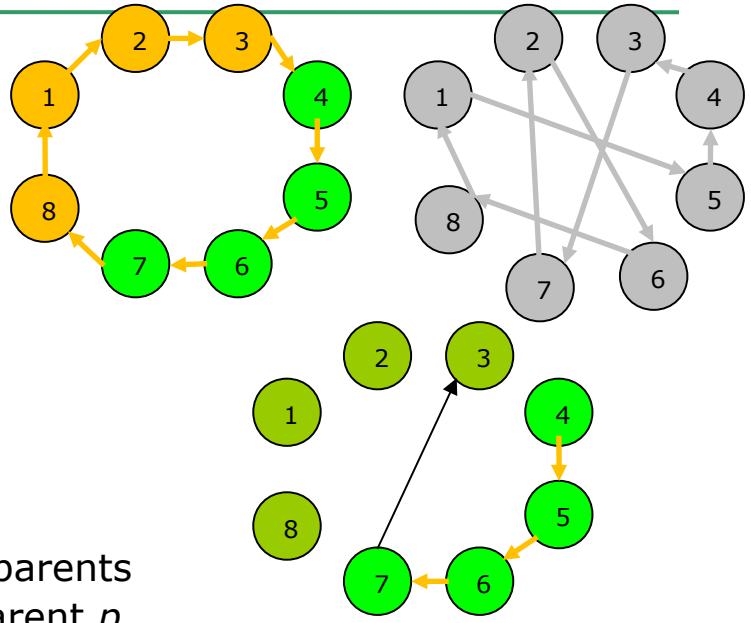


- Order crossover
  - Main idea
    - Offspring keep the order of genes from parents
    - Choose a substring of genes from the parent p_1
    - Copy the substring from p_1 into offspring d_1 (on corresponding positions)
    - Copy the genes of p_2 in offspring d_1:
      - Starting with the first position after sub-string
      - Respecting gene's order from p_2 and
      - Re-loading the genes from start (if the end of chromosome is reached)

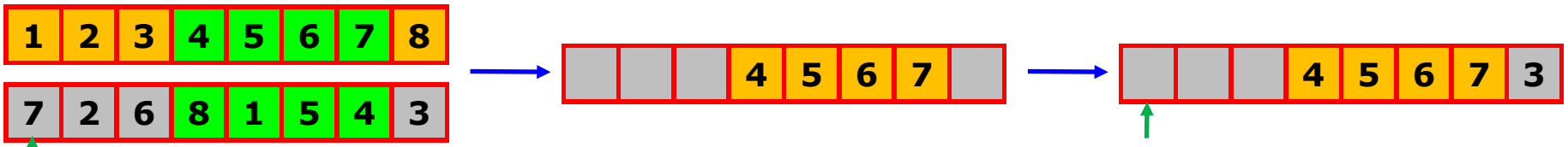


EAs - algorithm
Design – recombination (permutation representation)

- From 2 parent chromosomes
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ and $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- 2 offspring are obtained
  - $c_1=(g_1',g_2',\ldots,g_L')$ and $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - Where $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, for i=1,2,...,L

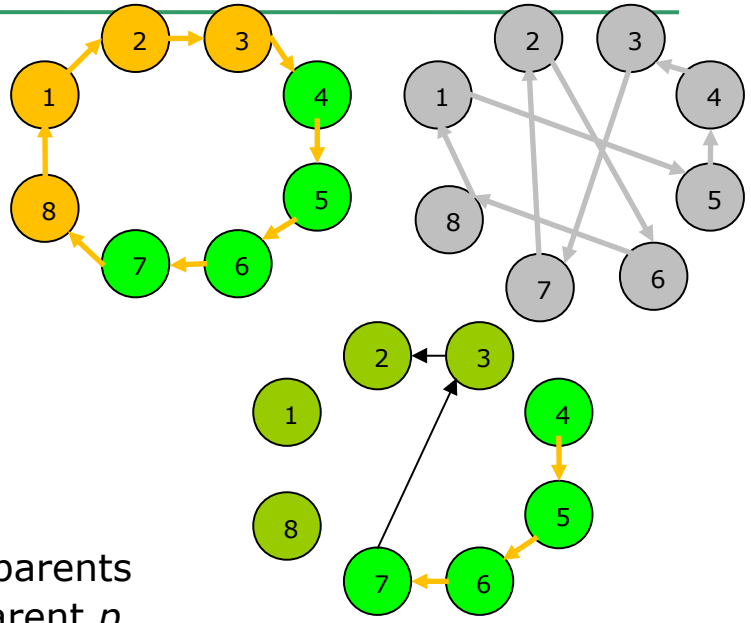


- Order crossover
  - Main idea
    - Offspring keep the order of genes from parents
    - Choose a substring of genes from the parent p_1
    - Copy the substring from p_1 into offspring d_1 (on corresponding positions)
    - Copy the genes of p_2 in offspring d_1:
      - Starting with the first position after sub-string
      - Respecting gene's order from p_2 and
      - Re-loading the genes from start (if the end of chromosome is reached)



EAs - algorithm
Design – recombination (permutation representation)

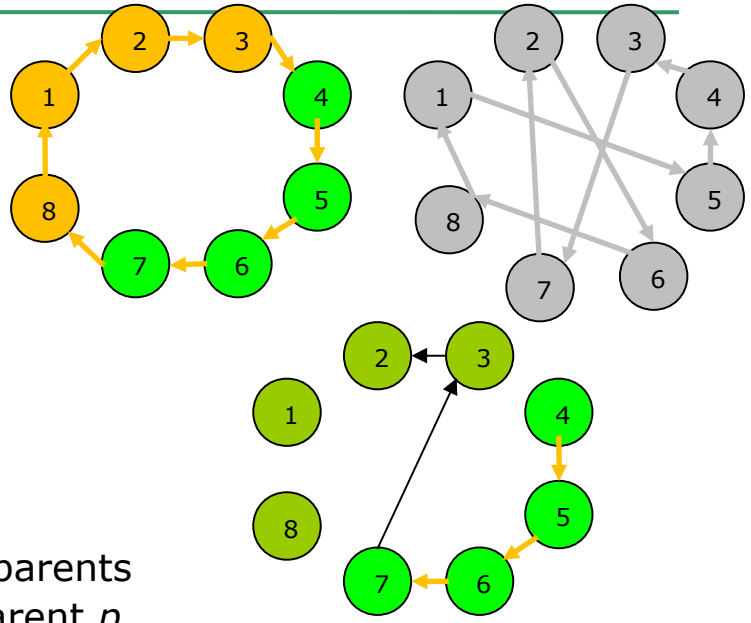
- From 2 parent chromosomes
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ and $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- 2 offspring are obtained
  - $c_1=(g_1',g_2',\ldots,g_L')$ and $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - Where $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, for i=1,2,...,L



- Order crossover
  - Main idea
    - Offspring keep the order of genes from parents
    - Choose a substring of genes from the parent p_1
    - Copy the substring from p_1 into offspring d_1 (on corresponding positions)
    - Copy the genes of p_2 in offspring d_1:
      - Starting with the first position after sub-string
      - Respecting gene's order from p_2 and
      - Re-loading the genes from start (if the end of chromosome is reached)

EAs - algorithm
Design – recombination (permutation representation)

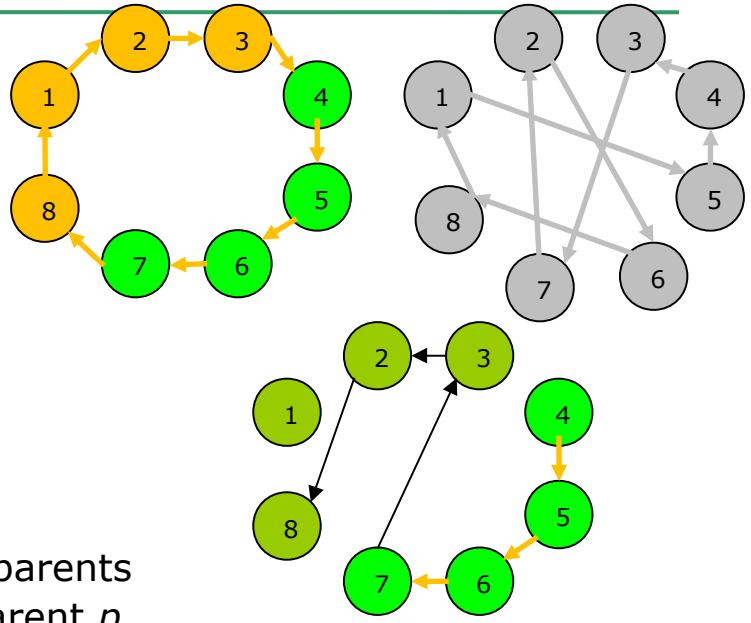
- From 2 parent chromosomes
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ and $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- 2 offspring are obtained
  - $c_1=(g_1',g_2',\ldots,g_L')$ and $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - Where $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, for i=1,2,...,L



- Order crossover
  - Main idea
    - Offspring keep the order of genes from parents
    - Choose a substring of genes from the parent p_1
    - Copy the substring from p_1 into offspring d_1 (on corresponding positions)
    - Copy the genes of p_2 in offspring d_1:
      - Starting with the first position after sub-string
      - Respecting gene's order from p_2 and
      - Re-loading the genes from start (if the end of chromosome is reached)

EAs - algorithm
Design – recombination (permutation representation)

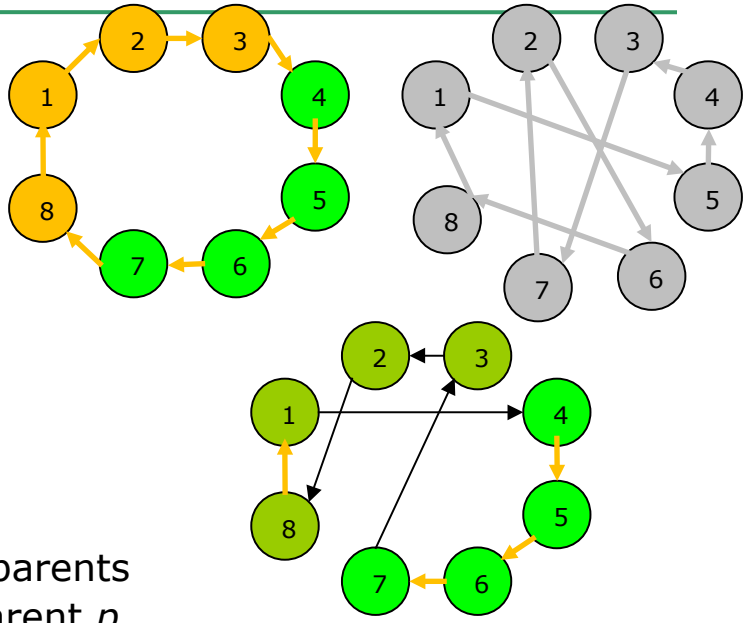
- From 2 parent chromosomes
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ and $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- 2 offspring are obtained
  - $c_1=(g_1',g_2',\ldots,g_L')$ and $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - Where $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, for i=1,2,…,L



- Order crossover
  - Main idea
    - Offspring keep the order of genes from parents
    - Choose a substring of genes from the parent p_1
    - Copy the substring from p_1 into offspring d_1 (on corresponding positions)
    - Copy the genes of p_2 in offspring d_1:
      - Starting with the first position after sub-string
      - Respecting gene's order from p_2 and
      - Re-loading the genes from start (if the end of chromosome is reached)

EAs - algorithm
Design – recombination (permutation representation)

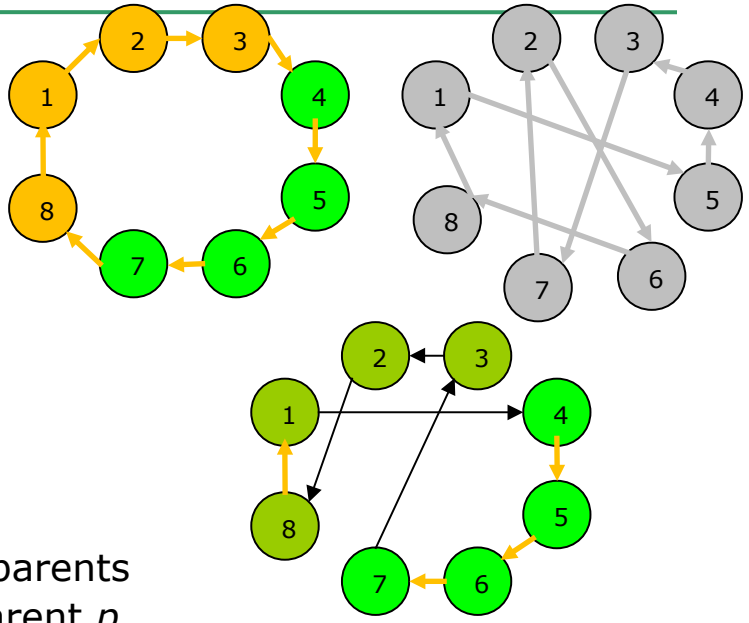
- From 2 parent chromosomes
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ and $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- 2 offspring are obtained
  - $c_1=(g_1',g_2',\ldots,g_L')$ and $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - Where $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, for i=1,2,…,L



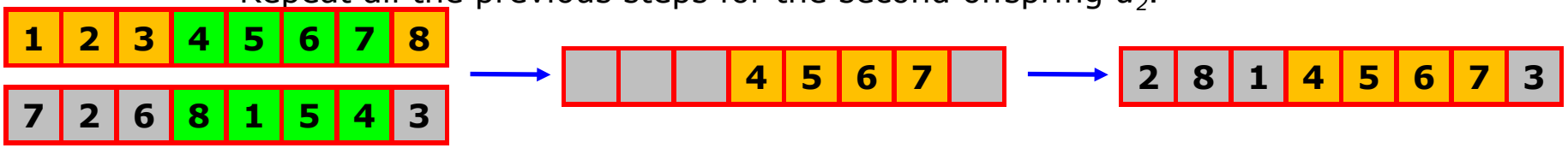
- Order crossover
  - Main idea
    - Offspring keep the order of genes from parents
    - Choose a substring of genes from the parent p_1
    - Copy the substring from p_1 into offspring d_1 (on corresponding positions)
    - Copy the genes of p_2 in offspring d_1:
      - Starting with the first position after sub-string
      - Respecting gene's order from p_2 and
      - Re-loading the genes from start (if the end of chromosome is reached)

EAs - algorithm
Design – recombination (permutation representation)

- From 2 parent chromosomes
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ and $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- 2 offspring are obtained
  - $c_1 =(g_1',g_2',\ldots,g_L')$ and $c_2 =(g_1'',g_2'',\ldots,g_L'')$,
  - Where $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, for i=1,2,…,L



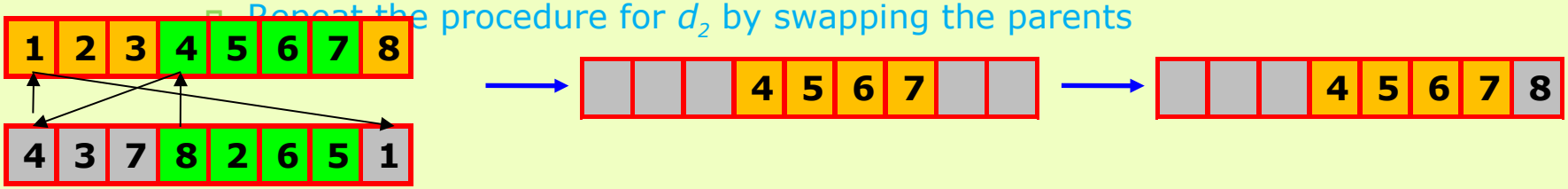
- Order crossover
  - Main idea
    - Offspring keep the order of genes from parents
    - Choose a substring of genes from the parent p_1
    - Copy the substring from p_1 into offspring d_1 (on corresponding positions)
    - Copy the genes of p_2 in offspring d_1:
      - Starting with the first position after sub-string
      - Respecting gene's order from p_2 and
      - Re-loading the genes from start (if the end of chromosome is reached)
    - Repeat all the previous steps for the second offspring d_2.



EAs - algorithm
Design – recombination (permutation representation)

- From 2 parent chromosomes
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ and $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- 2 offspring are obtained
  - $c_1=(g_1',g_2',\ldots,g_L')$ and $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - Where $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, for i=1,2,...,L

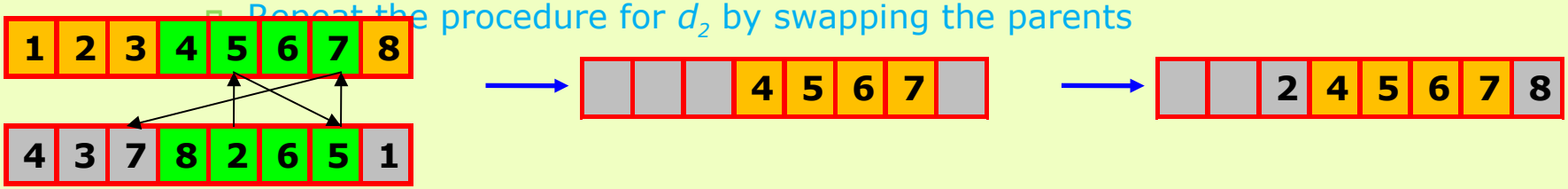
- *Partially mapped XO*
  - Main idea
    - Choose a substring of genes from parent p_1
    - Copy the sub-string into offspring d_1 (on corresponding positions)
    - Take elements *i* from substring of p_2 that do not belong to substring from p_1 and determine the element *j* that was copied instead of it from p_1
    - Put *i* in d_1 on position of *j* in p_2 (if that place is empty)
    - If the place of *j* in p_2 is already filled by element *k* in d_1, then *i* will be put in the position of *k* in p_2
    - All the other elements are copied from p_2 into d_1
    - Repeat the procedure for d_2 by swapping the parents



EAs - algorithm
Design – recombination (permutation representation)

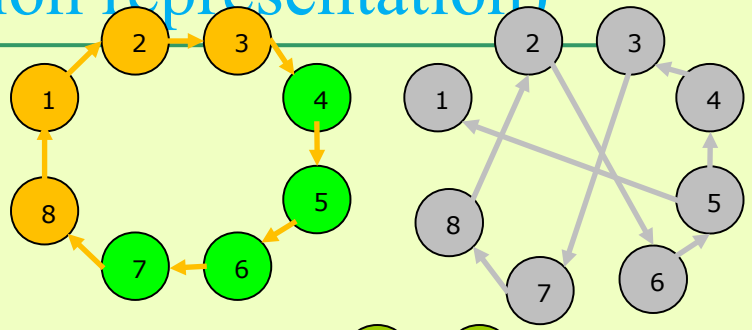
- From 2 parent chromosomes
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ and $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- 2 offspring are obtained
  - $c_1=(g_1',g_2',\ldots,g_L')$ and $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - Where $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, for i=1,2,...,L

- *Partially mapped XO*
  - Main idea
    - Choose a substring of genes from parent p_1
    - Copy the sub-string into offspring d_1 (on corresponding positions)
    - Take elements *i* from substring of p_2 that do not belong to substring from p_1 and determine the element *j* that was copied instead of it from p_1
    - Put *i* in d_1 on position of *j* in p_2 (if that place is empty)
    - If the place of *j* in p_2 is already filled by element *k* in d_1, then *i* will be put in the position of *k* in p_2
    - All the other elements are copied from p_2 into d_1
    - Repeat the procedure for d_2 by swapping the parents

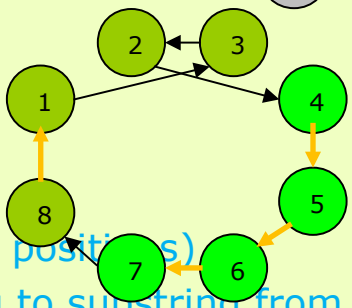


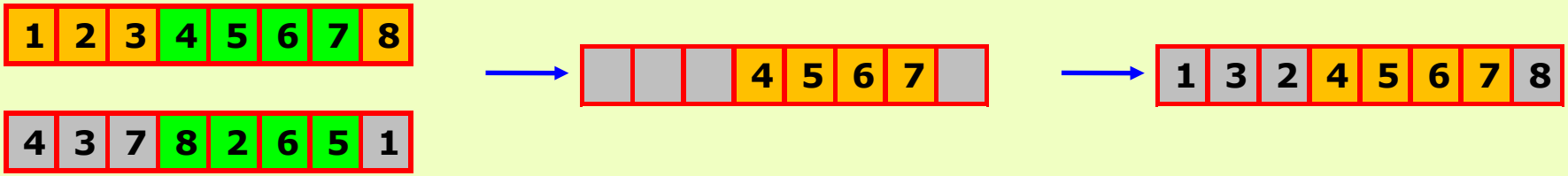
EAs - algorithm
Design – recombination (permutation representation)

- From 2 parent chromosomes
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ and $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- 2 offspring are obtained
  - $c_1=(g_1',g_2',\ldots,g_L')$ and $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - Where $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, for i=1,2,...,L



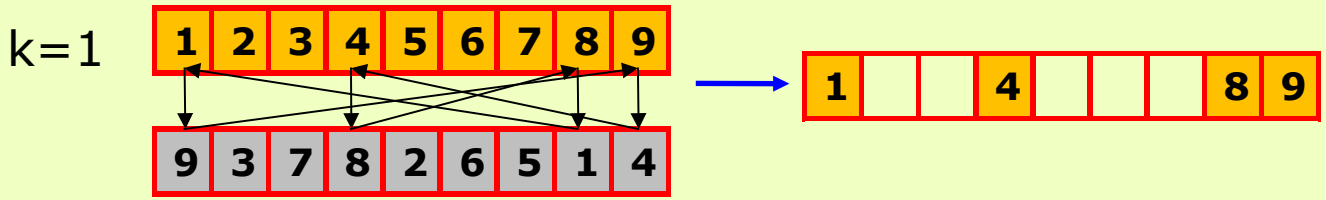
- *Partially mapped XO*
  - Main idea
    - Choose a substring of genes from parent p_1
    - Copy the sub-string into offspring d_1 (on corresponding positions)
    - Take elements *i* from substring of p_2 that do not belong to substring from p_1 and determine the element *j* that was copied instead of it from p_1
    - Put *i* in d_1 on position of *j* in p_2 (if that place is empty)
    - If the place of *j* in p_2 is already filled by element *k* in d_1, then *i* will be put in the position of *k* in p_2
    - All the other elements are copied from p_2 into d_1
    - Repeat the procedure for d_2 by swapping the parents





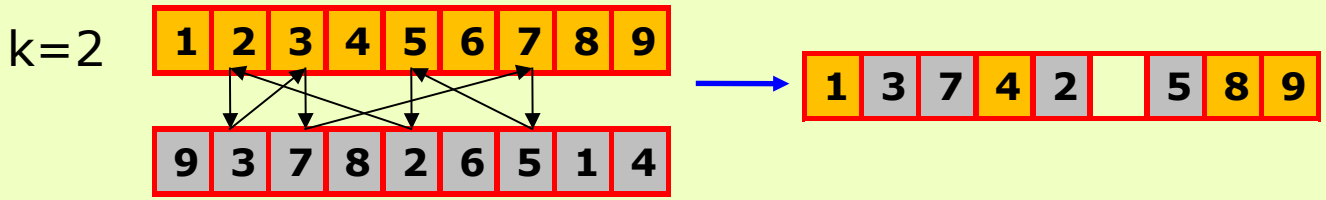
EAs - algorithm
Design – recombination (permutation representation)

- From 2 parent chromosomes
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ and $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- 2 offspring are obtained
  - $c_1=(g_1',g_2',\ldots,g_L')$ and $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - Where $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, for i=1,2,...,L
- Cycle crossover
  - Main idea
    1. Initialy, k = 1
    2. Create a cycle:
       - Add into the cycle the gene from position *k* from p_1 (g_k^1)
       - Take the gene of position *k* from p_2 (g_k^2)
       - Select the gene of p_1 whose value is equal to g_k^2 (g_r^1) and include it in the cycle
       - Take the gene of position *r* from p_2 (g_r^2)
       - Repeat the previous steps until the gene of position *k* from p_1 is considered
    3. Copy the genes of cycle into d_1 (by respecting the appearance positions in p_1)
    4. Increase *k* and compose an new cycle with the genes from p_2
    - Copy the genes of cycle into d_1 (by respecting the appearance positions in p_2)
    - Repeat steps 2-5 until $k = L$



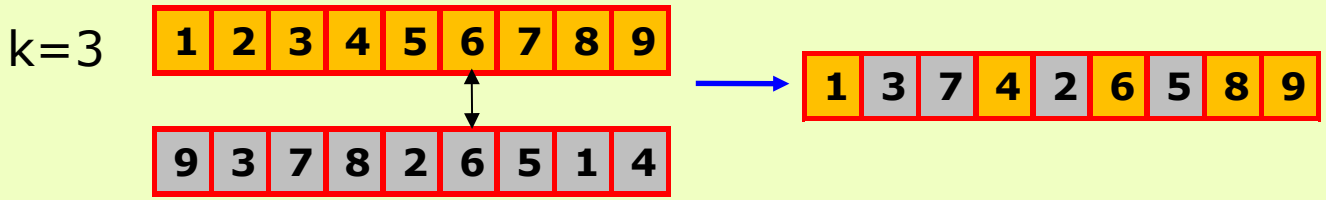
EAs - algorithm
Design – recombination (permutation representation)

- From 2 parent chromosomes
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ and $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- 2 offspring are obtained
  - $c_1=(g_1',g_2',\ldots,g_L')$ and $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - Where $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, for i=1,2,...,L
- Cycle crossover
  - Main idea
    1. Initialy, k = 1
    2. Create a cycle:
       - Add into the cycle the gene from position *k* from p_1 (g_k^1)
       - Take the gene of position *k* from p_2 (g_k^2)
       - Select the gene of p_1 whose value is equal to g_k^2 (g_r^1) and include it in the cycle
       - Take the gene of position *r* from p_2 (g_r^2)
       - Repeat the previous steps until the gene of position *k* from p_1 is considered
    3. Copy the genes of cycle into d_1 (by respecting the appearance positions in p_1)
    4. Increase *k* and compose an new cycle with the genes from p_2
    - Copy the genes of cycle into d_1 (by respecting the appearance positions in p_2)
    - Repeat steps 2-5 until $k = L$



EAs - algorithm
Design – recombination (permutation representation)

- From 2 parent chromosomes
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ and $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- 2 offspring are obtained
  - $c_1=(g_1',g_2',\ldots,g_L')$ and $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - Where $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, for i=1,2,...,L
- Cycle crossover
  - Main idea
    1. Initialy, k = 1
    2. Create a cycle:
       - Add into the cycle the gene from position *k* from p_1 (g_k^1)
       - Take the gene of position *k* from p_2 (g_k^2)
       - Select the gene of p_1 whose value is equal to g_k^2 (g_r^1) and include it in the cycle
       - Take the gene of position *r* from p_2 (g_r^2)
       - Repeat the previous steps until the gene of position *k* from p_1 is considered
    3. Copy the genes of cycle into d_1 (by respecting the appearance positions in p_1)
    4. Increase *k* and compose an new cycle with the genes from p_2
    - Copy the genes of cycle into d_1 (by respecting the appearance positions in p_2)
    - Repeat steps 2-5 until $k = L$



EAs - algorithm
Design – recombination (permutation representation)

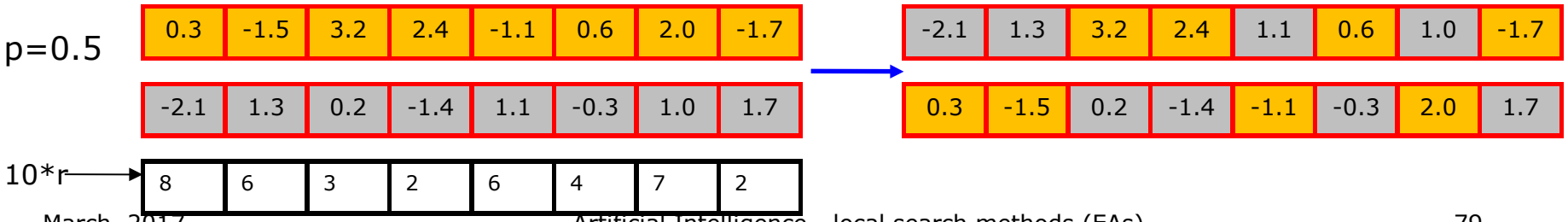
- From 2 parent chromosomes
  - $p_1=(g_1^1,g_2^1,...,g_L^1)$ and $p_2=(g_1^2,g_2^2,...,g_L^2)$
- 2 offspring are obtained
  - $c_1 =(g_1',g_2',...,g_L')$ and $c_2 =(g_1'',g_2'',...,g_L'')$,
  - Where $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, for i=1,2,...,L

- Edge-based crossover
  - See *Whitley, Darrell, Timothy Starkweather, D'Ann Fuquay (1989). "Scheduling problems and traveling salesman: The genetic edge recombination operator".International Conference on Genetic Algorithms. pp. 133–140* link

EAs - algorithm
Design – recombination (real representation)

- From 2 parent chromosomes
  - $p_1=(g_1^1,g_2^1,...,g_L^1)$ and $p_2=(g_1^2,g_2^2,...,g_L^2)$
- 2 offspring are obtained
  - $c_1 =(g_1',g_2',...,g_L')$ and $c_2 =(g_1'',g_2'',...,g_L'')$,
  - Where $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, for i=1,2,...,L

- Discrete crossover
  - Main idea
    - Each gene offspring is taken (by the same probability, *p = 0.5*) from one of the parents
    - Similarly to uniform crossover for binary/integer representation
    - The absolute values of genes are not changed (no new information is created)



EAs – algorithm
Design – recombination (real representation)

- From 2 parent chromosomes
  - $p_1=(g_1^1,g_2^1,...,g_L^1)$ and $p_2=(g_1^2,g_2^2,...,g_L^2)$
- 2 offspring are obtained
  - $c_1=(g_1',g_2',...,g_L')$ and $c_2=(g_1'',g_2'',...,g_L'')$,
  - Where $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, for i=1,2,...,L

- Arithmetic crossover
  - Main idea
    - Create offspring between parents → arithmetic crossover
      - $z_i = \alpha x_i + (1 - \alpha) y_i$ where α : $0 \leq \alpha \leq 1$.
    - Parameterl α can be:
      - Constant → uniform arithmetic crossover
      - Variable → eg. Depends on the age of population
      - Random → generated for each new XO that is performed
    - New values of a gene can appear
  - Typology
    - Singular arithmetic crossover
    - Simple arithmetic crossover
    - Complete arithmetic crossover

EAs – algorithm
Design – recombination (real representation)

- From 2 parent chromosomes
  - $p_1=(g_1^1,g_2^1,...,g_L^1)$ and $p_2=(g_1^2,g_2^2,...,g_L^2)$
- 2 offspring are obtained
  - $c_1=(g_1',g_2',...,g_L')$ and $c_2=(g_1'',g_2'',...,g_L'')$,
  - Where $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, for i=1,2,...,L

- Singular arithmetic crossover
  - Choose one gene from two parents (of the same position k) and combine them
    - $g_k' = \alpha g_k^1 + (1-\alpha) g_k^2$
    - $g_k'' = (1-\alpha) g_k^1 + \alpha g_k^2$
  - The rest of genes are unchanged
    - $g_i' = g_i^1$
    - $g_i'' = g_i^2$, for i = 1,2,...,L and $i \neq k$

[LI,LS] = [-2.5, +3]
k=3
α = 0.6

0.3	-1.5	3.2	2.4	-1.1	0.6	2.0	-1.7
-2.1	1.3	0.2	-1.4	1.1	-0.3	1.0	1.7

0.6*3.2+(1-0.6)*0.2=2.0

(1-0.6)*3.2+0.6*0.2=1.4

0.3	-1.5	2.0	2.4	-1.1	0.6	2.0	-1.7
-2.1	1.3	1.4	-1.4	1.1	-0.3	1.0	1.7

EAs – algorithm
Design – recombination (real representation)

- From 2 parent chromosomes
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ and $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- 2 offspring are obtained
  - $c_1=(g_1',g_2',\ldots,g_L')$ and $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - Where $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, for i=1,2,...,L

- Simple arithmetic crossover
  - Select a position k and combine all the genes after that position
    - $g_i' = \alpha g_i^1 + (1-\alpha)g_i^2$
    - $g_i'' = (1-\alpha)g_i^1 + \alpha g_i^2$, for $i=k, k+1, \ldots, L$
  - Genes from positions < k rest unchanged
    - $g_i' = g_i^1$
    - $g_i'' = g_i^2$, for $i = 1,2,\ldots,k-1$

[LI,LS] = [-2.5, +3]
k=6
α = 0.6

0.3	-1.5	3.2	2.4	-1.1	0.6	2.0	-1.7
-2.1	1.3	0.2	-1.4	1.1	-0.3	1.0	1.7

0.6*0.6+(1-0.6)*(-0.3)=0.24

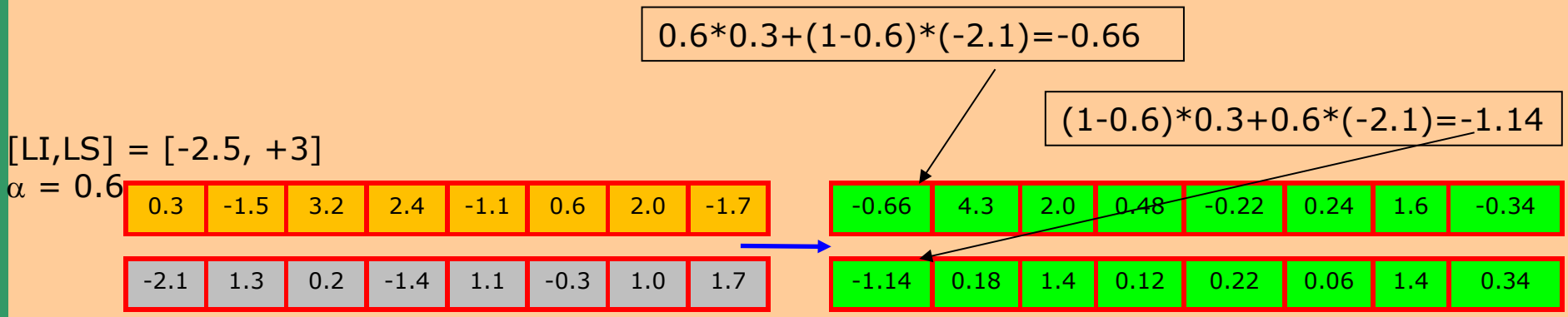
(1-0.6)*0.6+0.6*(-0.3)=0.06

0.3	-1.5	3.2	2.4	-1.1	0.24	1.6	-0.34
-2.1	1.3	0.2	-1.4	1.1	0.06	1.4	0.34

EAs – algorithm
Design – recombination (real representation)

- From 2 parent chromosomes
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ and $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- 2 offspring are obtained
  - $c_1=(g_1',g_2',\ldots,g_L')$ and $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - Where $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, for i=1,2,...,L

- Complete arithmetic crossover
  - All of the genes are combined
    - $g_i' = \alpha g_i^1 + (1-\alpha) g_i^2$
    - $g_i'' = (1-\alpha) g_i^1 + \alpha g_i^2$, for i=1,2,...,L



EAs – algorithm
Design – recombination (real representation)

- From 2 parent chromosomes
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ and $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- 2 offspring are obtained
  - $c_1=(g_1',g_2',\ldots,g_L')$ and $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - Where $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, for i=1,2,...,L

- Geometric crossover
  - Main idea
    - Each gene of an offspring represents the product between parent's genes, each of them by a given exponent ω and 1-ω, respectively (where ω is a real positive number <= 1)
    - $g_i' = (g_i^1)^{\omega} (g_i^2)^{1-\omega}$
    - $g_i'' = (g_i^1)^{1-\omega} (g_i^2)^{\omega}$

$0.3^{0.7}+2.1^{1-0.7}=1.68$

$0.3^{1-0.7}+2.1^{0.7}=2.38$

[LI,LS] = [-2.5, +3]

ω = 0.7

0.3	1.5	3.2	2.4	1.1	0.6	2.0	1.7
2.1	1.3	0.2	1.4	1.1	0.3	1.0	1.7

1.68	2.41	2.87	2.95	2.10	1.40	2.62	2.62
2.38	2.33	1.74	2.57	2.10	1.29	2.23	2.62

EAs – algorithm
Design – recombination (real representation)

- From 2 parent chromosomes
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ and $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- 1 offspring is obtained
  - $c_1=(g_1',g_2',\ldots,g_L')$
  - Where $g_i^1, g_i' \in [LI_i, LS_i]$, for i=1,2,...,L

- *Blend crossover – BLX*
  - Main idea
    - A single offspring is created
    - Offspring's genes are randomly generated from $[Min_i-I*a, Max_i+I*a]$ range, where:
      - $Min_i = min\{g_i^1, g_i^2\}$, $Max_i = max\{g_i^1, g_i^2\}$
      - $I = Max - Min$, a – parameter from [0,1]

[LI,LS] = [-2.5, +3]
a = 0.7

0.3	1.5	3.2	2.4	1.1	0.6	2.0	1.7
2.1	1.3	0.2	1.4	1.1	0.3	1.0	1.7

→

1.25	1.45	-1.11	2.37	1.10	0.11	0.70	1.70

Min	0.3	1.3	0.2	1.4	1.1	0.3	1.0	1.7
Max	2.1	1.5	3.2	2.4	1.1	0.6	2.0	1.7
I	0.8	0.2	3.0	1.0	0	0.3	1.0	0.0

Min-Ia	-0.26	1.16	-1.90	0.70	1.10	0.09	0.30	1.70
Max+Ia	2.66	1.50	3.20	2.40	1.10	0.60	2.00	1.70

EAs – algorithm Design – recombination (real representation)

- From 2 parent chromosomes
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ and $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- 2 offspring are obtained
  - $c_1=(g_1',g_2',\ldots,g_L')$ and $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - Where $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, for $i=1,2,\ldots,L$

- Simulated binary crossover
  - Main idea
    - Each gene of an offspring is a combination of parent's genes

$$d_1 = \frac{p_1+p_2}{2} - \beta\frac{p_2-p_1}{2}, \quad d_2 = \frac{p_1+p_2}{2} + \beta\frac{p_2-p_1}{2}$$

    - Such as the two properties of n-cutting point XO to be respected (for binary representation)
      - Average of parent values = average of offspring values
      - Probability of a spread factor $\beta \approx 1$ is greater to any other factor

EAs – algorithm Design – recombination

- Multiple recombination
  - Based on the value's frequencies from parents (general uniform XO)
  - Based on segmentation and crossover (general XO with diagonal cutting points)
  - Based on numeric operations that are specific to real values (XO based on gravity center, general arithmetic XO)

EAs – algorithm
Design – recombination or mutation?

- Intense debates
  - Questions:
    - Which is the best operator?
    - Which is the most necessary operator?
    - Which is the most important operator?
  - Answers:
    - Depend on problem, but,
    - In general, is better to use both operators
    - Each of them having another role (purpose).
    - EAs with mutation only are possible, but EAs with crossover only are not possible
- Search aspects:
  - Exploration → discovering promising regions in the search space (accumulating useful information about the problem)
  - Exploitation → optimising in a promising region of the search space (by using the existent information)
  - Cooperation and competition mut exist between these 2 aspects
- Recombination
  - Exploitation operator → performs a large jump into a region somewhere between the regions associated to parents
    - Effects of exploitation decrease while AE is converging
  - Binary/n-ary operator that can combine information from 2/more parents
  - Operator that does not change the frequency of values from chromosome at the population level
- Mutation
  - Exploration operator → performs small random diversions, remaining in a neighbourhood of parent
    - Local optima escape
  - Operator that can introduce new genetic information
  - Operator that change the frequency of values from chromosome at the population level

EAs – algorithm
Design – stop condition

- Choosing a stop condition
  - An optimal solution was found
  - The physical resources were ended
    - A given number of fitness evaluation has been performed
  - The user resources (time, patience) were ended
    - Several generation without improvements have been born

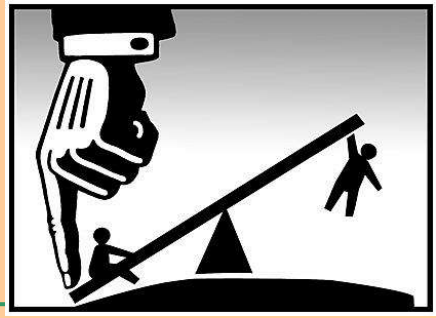
EAs – algorithm Evaluation

- Performance evaluation of an EA
  - After more runs
    - Statistical measures are computed
      - Average of solutions
      - Median of solutions
      - Best solution
      - Worst solution
      - Standard deviation of solutions – for comparisons
  - The number of independent runs must be large enough

EAs

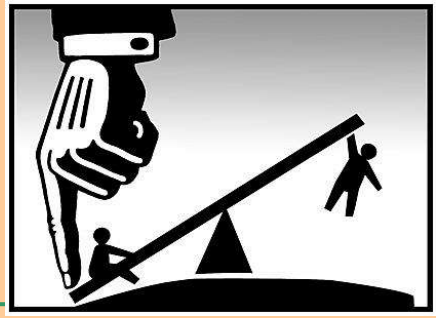
- Analyse of complexity
  - The most costly part → fitness evaluation

EAs



- Advantages
  - AEs have a general sketch for all the problems
    - Only
      - representation
      - fitness function
    - are changed
  - AEs are able to give better results than classical optimisation methods because
    - They do not require linearization
    - They are not based on some presumptions
    - They do not ignore some possible solutions
  - AEs are able to explore more possible solutions than human can

AEs



- Disadvantages
  - Large running time

AEs

- Applications
  - Vehicle design
    - Material composition
    - Vehicle shape
  - Engineering design
    - Structural and organisational optimisation of constructions (buildings, robots, satellites, turbines)
  - Robotics
    - Design and components optimisation
  - Hardware evolution
    - Digital circuits optimisation
  - Telecommunication optimisation
  - Cross-word game generation
  - Biometric inventions (inspired by natural architectures)
  - Traffic and transportation routing
  - PC games
  - Cryptography
  - Genetics
  - Chemical analyse of kinematics
  - Financial and marketing strategies

AEs

- Typology
  - Evolutionary strategies
  - Evolutionary programming
  - Genetic algorithms
  - Genetic programming

Next lecture

A. Short introduction in Artificial Intelligence (AI)

A. **Solving search problems**
   A. **Definition of search problems**
   B. **Search strategies**
      A. Uninformed search strategies
      B. Informed search strategies
      C. **Local search strategies (**Hill Climbing, Simulated Annealing, Tabu Search, Evolutionary algorithms, **PSO, ACO**)
      D. Adversarial search strategies

C. Intelligent systems
   A. Rule-based systems in certain environments
   B. Rule-based systems in uncertain environments (Bayes, Fuzzy)
   C. Learning systems
      A. Decision Trees
      B. Artificial Neural Networks
      C. Support Vector Machines
      - Evolutionary algorithms
   D. Hybrid systems

Next lecture – Useful information

- Chapter 16 of *C. Groşan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- *James Kennedy, Russel Eberhart, Particle Swarm Optimisation*, *Proceedings of IEEE International Conference on Neural Networks*. **IV**. pp. 1942–1948, 1995 (04_ACO_PSO/PSO_00.pdf)
- *Marco Dorigo, Christian Blum, Ant colony optimization theory: A survey, Theoretical Computer Science 344 (2005) 243 – 27* (04_ACO_PSO/Dorigo05_ACO.pdf)

- Presented information have been inspired from different bibliographic sources, but also from past AI lectures taught by:
  - PhD. Assoc. Prof. Mihai Oltean – www.cs.ubbcluj.ro/~moltean
  - PhD. Assoc. Prof. Crina Groşan - www.cs.ubbcluj.ro/~cgrosan
  - PhD. Prof. Horia F. Pop - www.cs.ubbcluj.ro/~hfpop