Wrocław University
of Science and Technology

AKADEMIA INNOWACYJNYCH ZASTOSOWAŃ TECHNOLOGII CYFROWYCH (AI TECH)

Genetic Algorithms
Basics and why does it work?

Michał Przewoźniczek
Department of Systems and Computer Networks
Wroclaw University of Science and Technology

Rzeczpospolita
Polska

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego,
Program Operacyjny Polska Cyfrowa na lata 2014-2020,
Oś Priorytetowa nr 3 "Cyfrowe kompetencje społeczeństwa" Działanie nr 3.2 "Innowacyjne rozwiązania na rzecz aktywizacji cyfrowej"
Tytuł projektu: „Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”

Optimization problems

Definition

- The obejctive function for n-dimensional optimization problem:

$$f: D_f \subseteq R^n \rightarrow R$$

where: D_f – solution search space

- Obejctive: find the best $\vec{x}^*$

$$\vec{x}^* = \arg\min_{\vec{x} \in D_{\vec{x}} \subseteq D_f} f(\vec{x}) \quad \text{or} \quad \vec{x}^* = \arg\max_{\vec{x} \in D_{\vec{x}} \subseteq D_f} f(\vec{x})$$

where: $D_{\vec{x}}$ – the feasible set

Optimization problems

Continuous search space

- The domain of the optimized function:

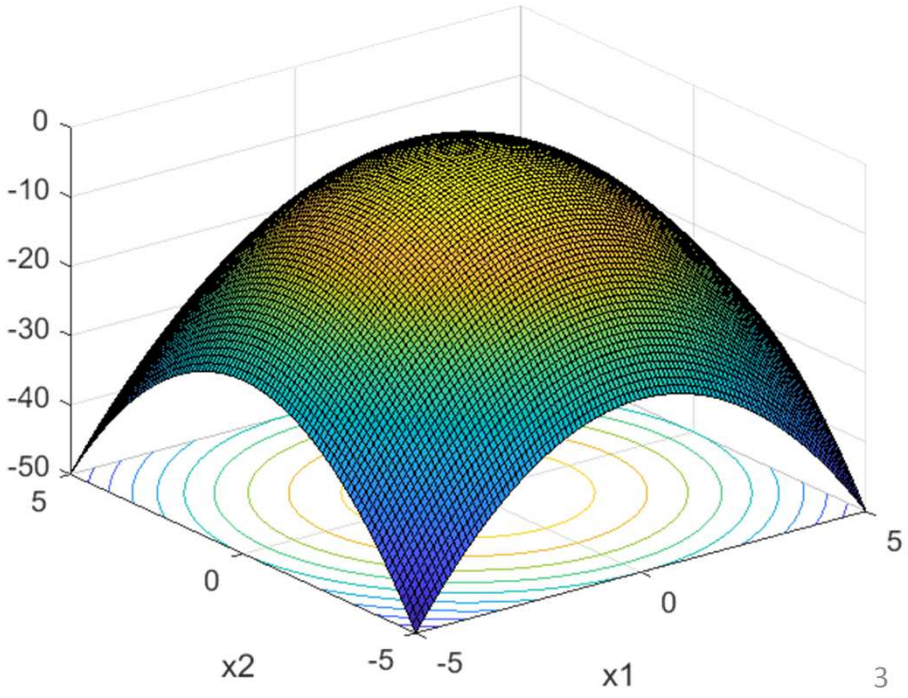
$$D_f \subseteq R^n$$

- Example: two-dimensional sphere function

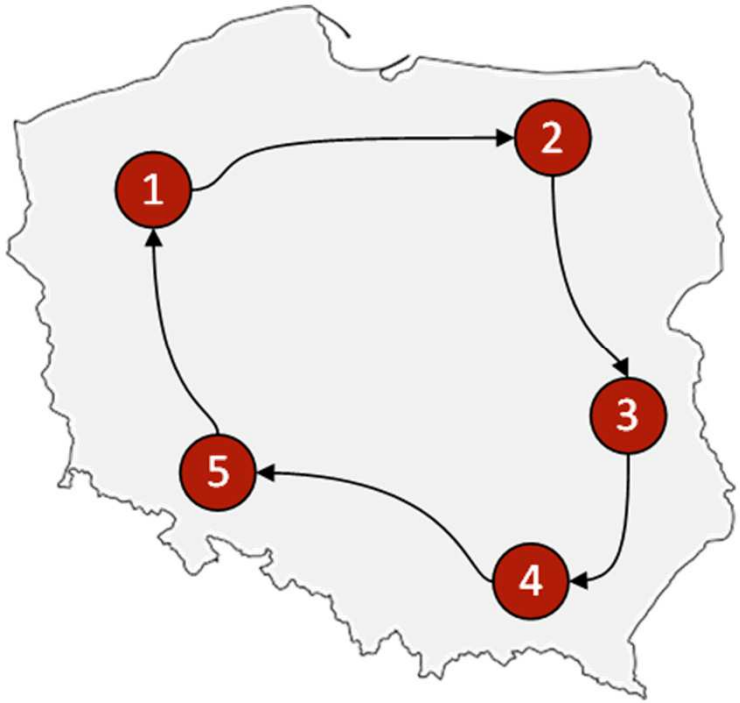
$\vec{x} = [x_1, x_2]$
$f: D_f = R^2 \rightarrow R$
$f(\vec{x}) = -x_1^2 - x_2^2$

We limit the search space to

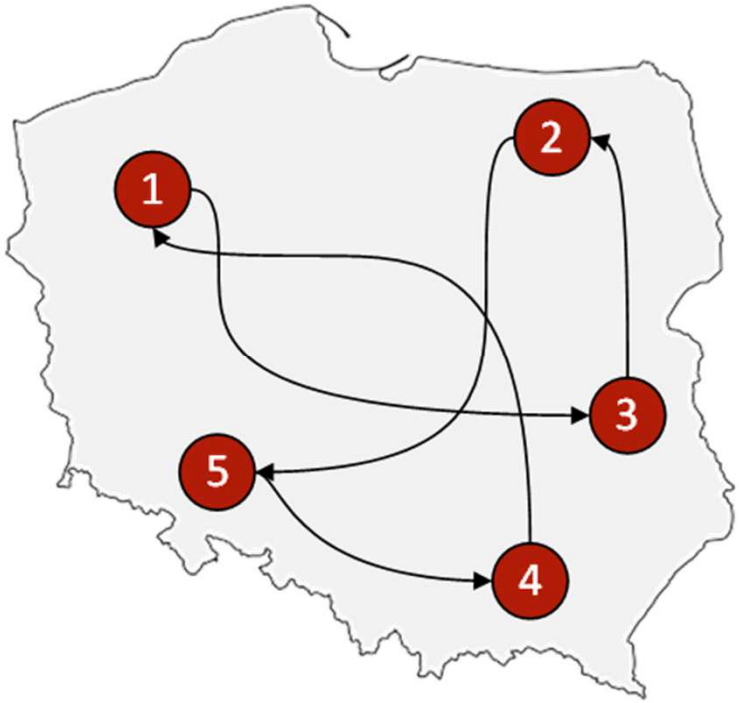
$D_{\vec{x}} = [-5, 5]^2 \subseteq D_f$



Travelling Salesman Problem



$[1, 2, 3, 4, 5]$



$[1, 3, 2, 5, 4]$

Optimization – what is it?

Traveling Salesman Problem

- Single solution:

$$\vec{x} = [x_1, x_2, x_3, x_4, x_5]$$

$$\forall_{i \in \{1,\dots,5\}} x_i \in D_{x_i} = \{1, \dots, 5\}$$

- The set of **all** solutions:

$$D_f = D_{x_1} \times D_{x_2} \times D_{x_3} \times D_{x_4} \times D_{x_5}$$

- The set of feasible solutions:

$$D_{\vec{x}} = \left\{\vec{x}: \forall_{i \in \{1,\dots,5\}} \forall_{j \in \{1,\dots,5\}-\{i\}} x_i \in D_{x_i} \wedge x_j \in D_{x_j} \wedge x_i \neq x_j\right\}$$

Problem nature reminder

- Travelling Salesman Problem (TSP) – ***combinatorial*** *in nature* **<- this we are going to solve today**
  - We want to exchange solution fragments
  - For instacne – the „good” city sequences

- Hill – ***topological*** *in nature* **<- but there will be some examples considering the continuous search spaces**
  - We want to search for the better solution in the neighbourhood of the best solutions found that far
  - We shift „slightly left”, or „ slightly right”

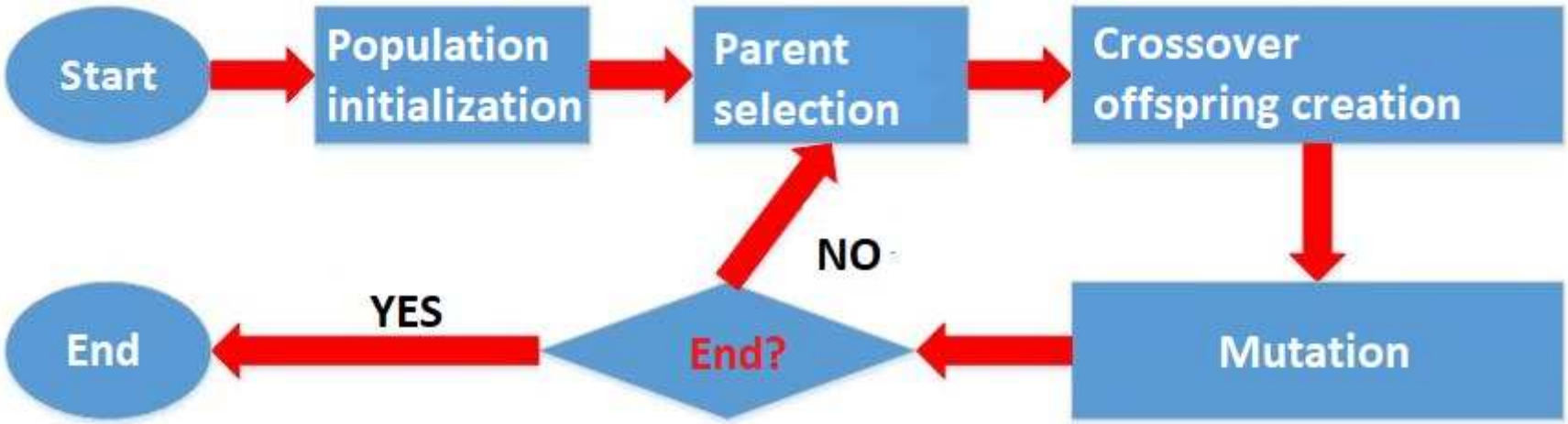
Genetic Algorithms
Where did it come from?

- How about simulating the evolution?
- Objective: investigate (increase the understanding) of the evolution process

„Adaptation in Natural and Artificial Systems"
John Henry Hollad, 1975

Genetic Algorithms

Basics



Genetic Algorithms

Basics

Main features of the evolution:

- **Individual** → **genotyp** → defines the features of an individual
- Selection → better fitting individuals:
  - Higher chance to survive
  - Higher chance for offspring
- **offspring** → genes from parents → **krzyżowanie** (ang. *corssover*)
- **Mutation** → random (slight) modification of the genotype

Genetic Algorithms

example in continuous space

- Fitness → function value
- Solutions → individual → genotype

  2 real numbers from the interval [0; 5]
- **Corssover**:

1st offspring: 1st gene of ***prent 1*** + 2nd gene of ***parent 2***

2nd offspring: 1st gene of ***prent 2*** + 2nd gene of ***parent 1***

Mum: <2.5; 2.8> Dad: <1.2; 4.8>

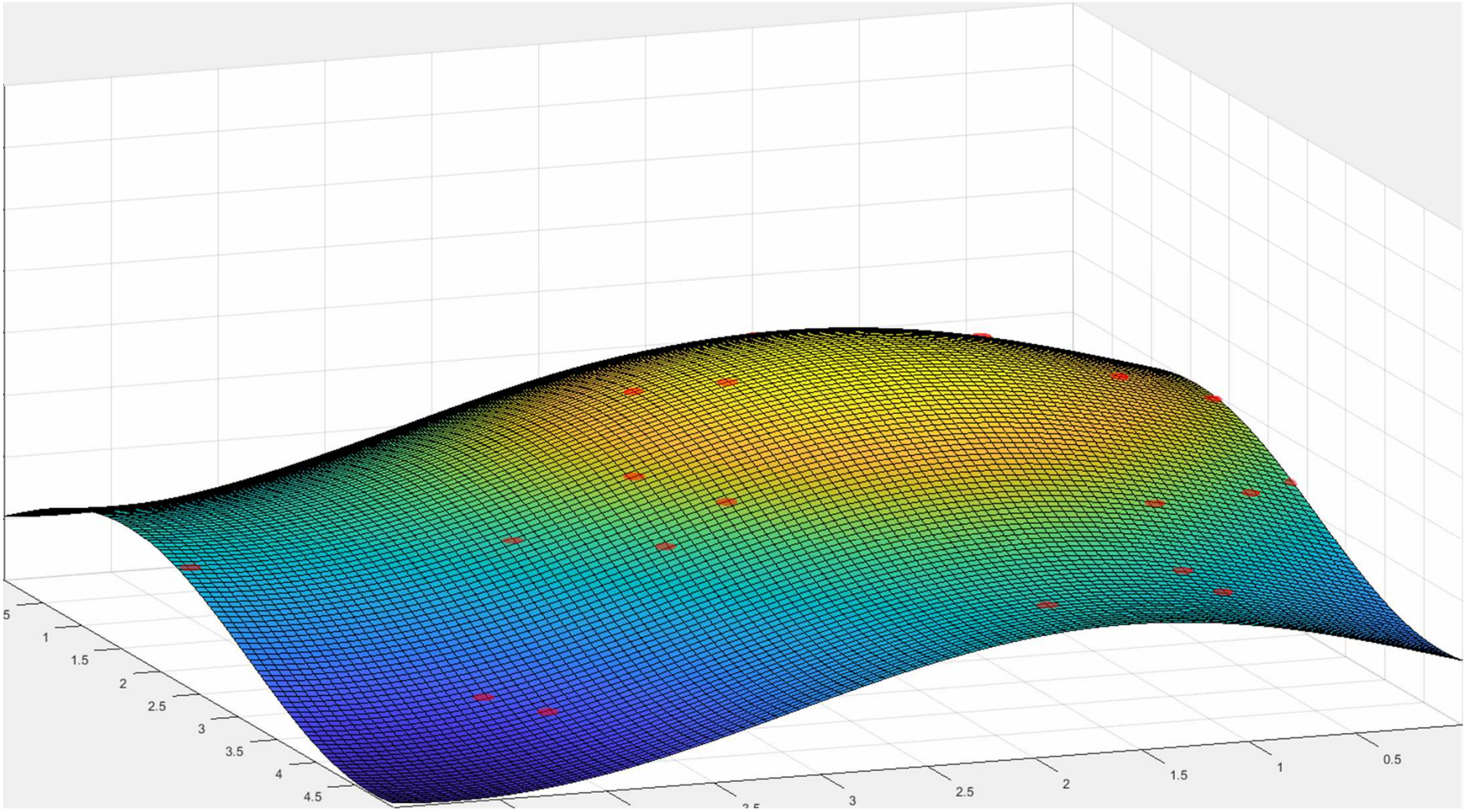
Doughter: <2.5; 4.8> Son: <1.2; 2.8>

- Mutation:

  add random value from the interval [-0.15; 0.15]

Wrocław University
of Science and Technology

Genetic Algorithms
example in continuous space



Genetic Algorithms
example in continuous space

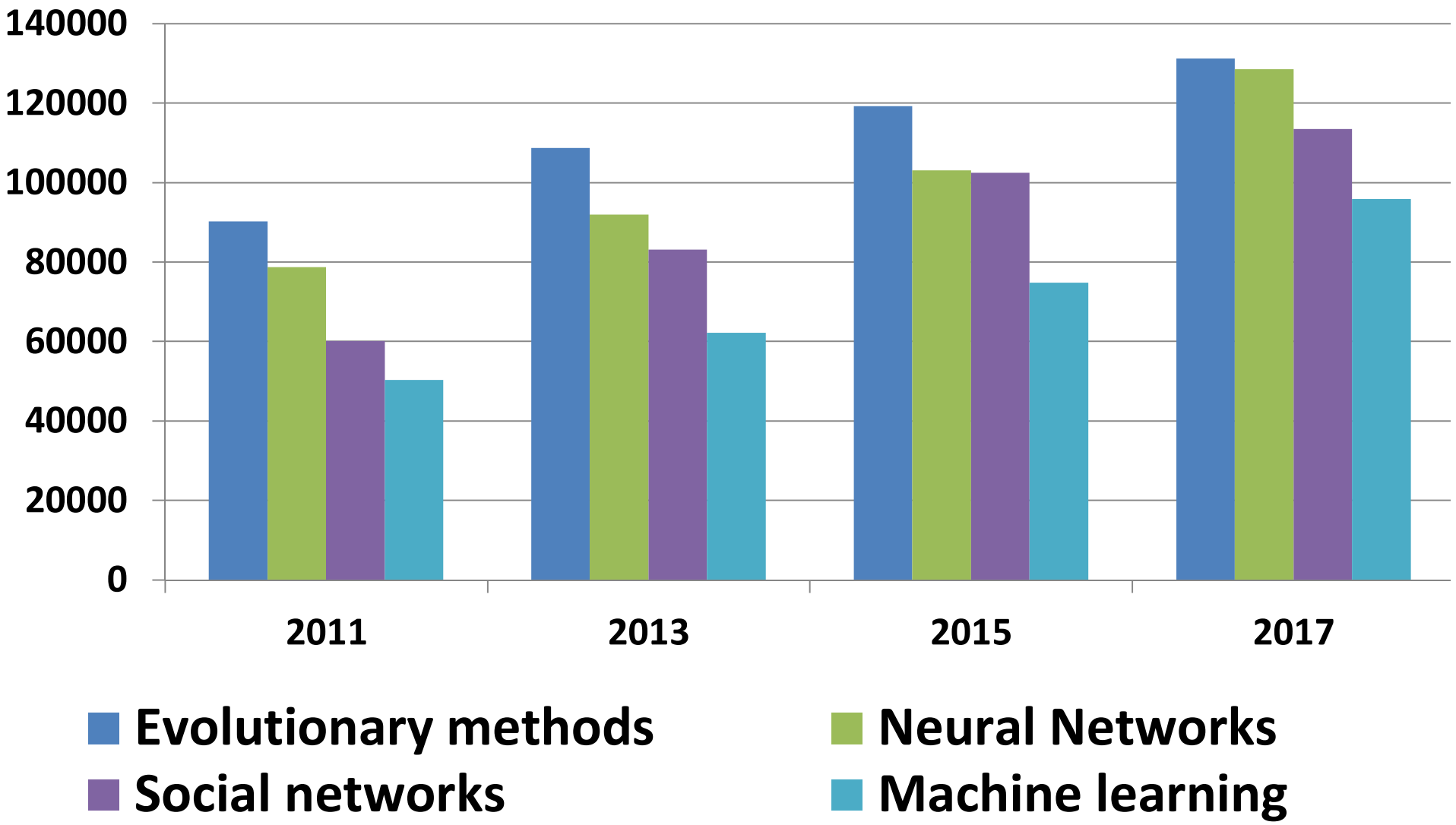
- **Evolution simulation = optimizer**
- Consequences
  - Precise evolutiona simulation (evolution itself, animal behaviour, etc.) - **IRRELEVANT**
  - **Create an optimizer, that solves the problem**

Wrocław University
of Science and Technology

Genetic Algorithms classification

- Artificial inteligence
  - ***Evolutionary Computation***
  - *Machine Learning*
  - *Expert systems*
  - *Image processing*
  - *Social netoworks analysis*
  - other…

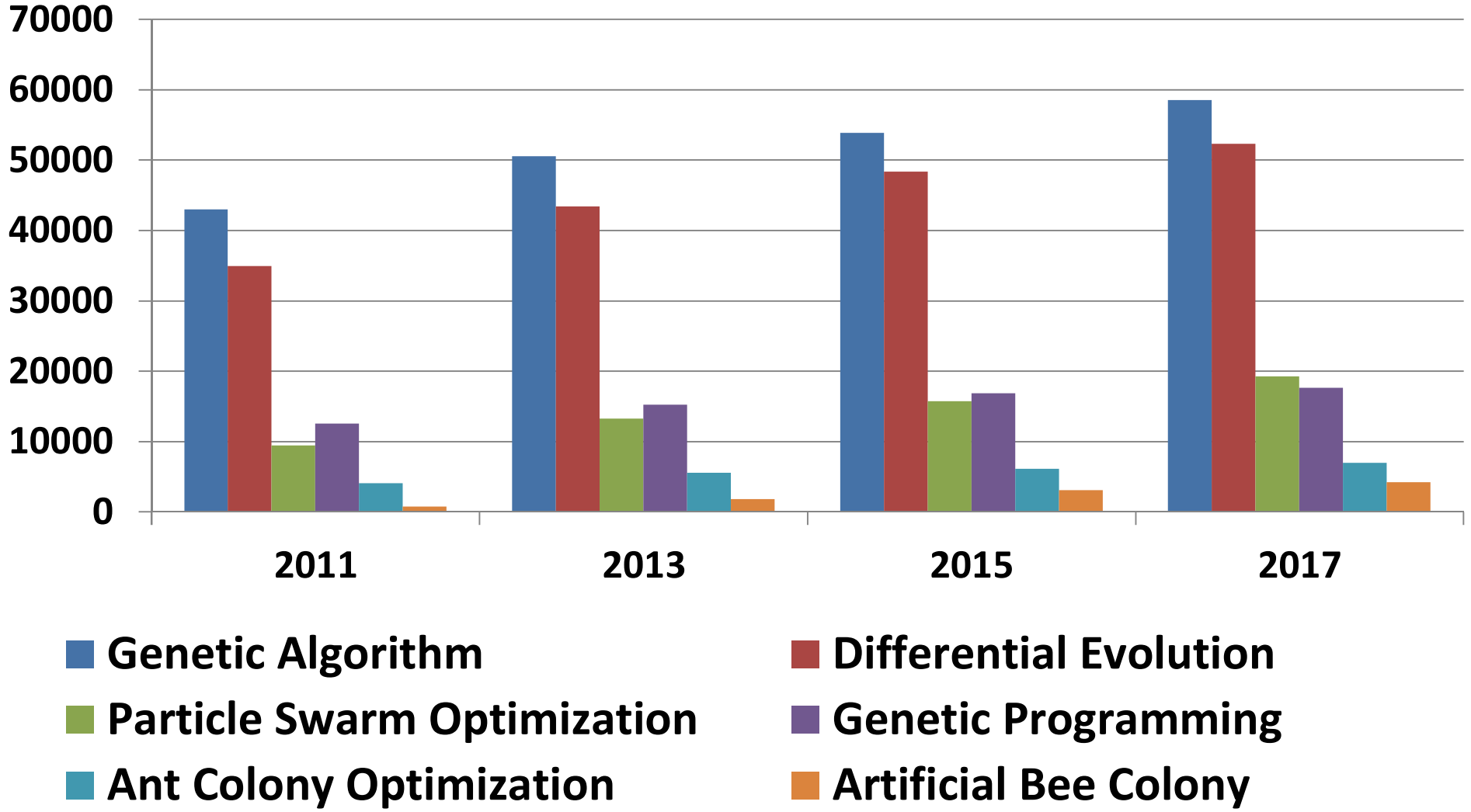
Wrocław University
of Science and Technology



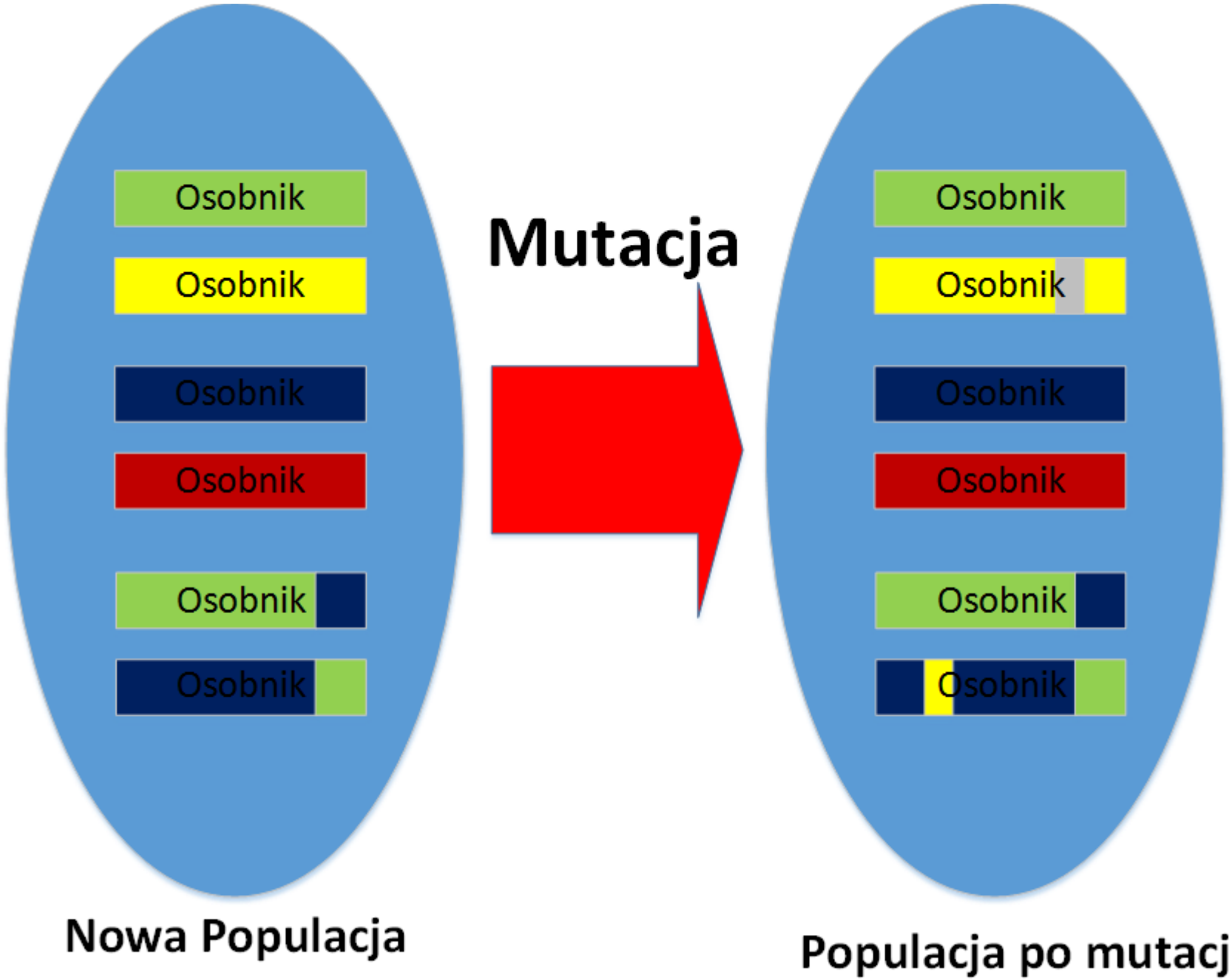
Research paper number (Scopus)

Genetic Algorithms classification

- *Genetic Algorithms*
- *Evolutionary strategies*
- *Genetic Programming*
- *Differential Evolution*
- *Particle Swarm Optimization*
- *Ant Colony Optimization*
- *Artificial Bee Colony*
- *Evolutionary programming*
- Other…



Research paper number (Scopus)


Osobnik
Osobnik
Osobnik
Osobnik
Osobnik
Osobnik
Mutacja
Osobnik
Osobnik
Osobnik
Osobnik
Osobnik
Osobnik
Nowa Populacja
Populacja po mutacji

Genetic Algorithms key elements

- Selection – how and why?
  - Give weaker individual a Chance (but do not cross the line)
  - Too strong selection pressure → **preconvergence**

    Intuition: weaker individual **may have some positive features too**
  - Too weak selectiono pressure → **no** convergence

    Intuition: in general we prefer **better-fitting** individuals
- Main selection techniques:
  - Roulette wheel
  - Tournament

Genetic Algorithms
Roulette wheel

- Chance for selection – proportional to fitness

$$p_i(x_i) = \frac{fitness(x_i)}{\sum_{j=1}^{size} fitness(x_j)}$$

where:

- x_i – *i*th individual
- $p_i(x_i)$ – the probability of choosing the *i*th individual
- $fitness(x_i)$ - fitness of the *i*th individual
- size – population size

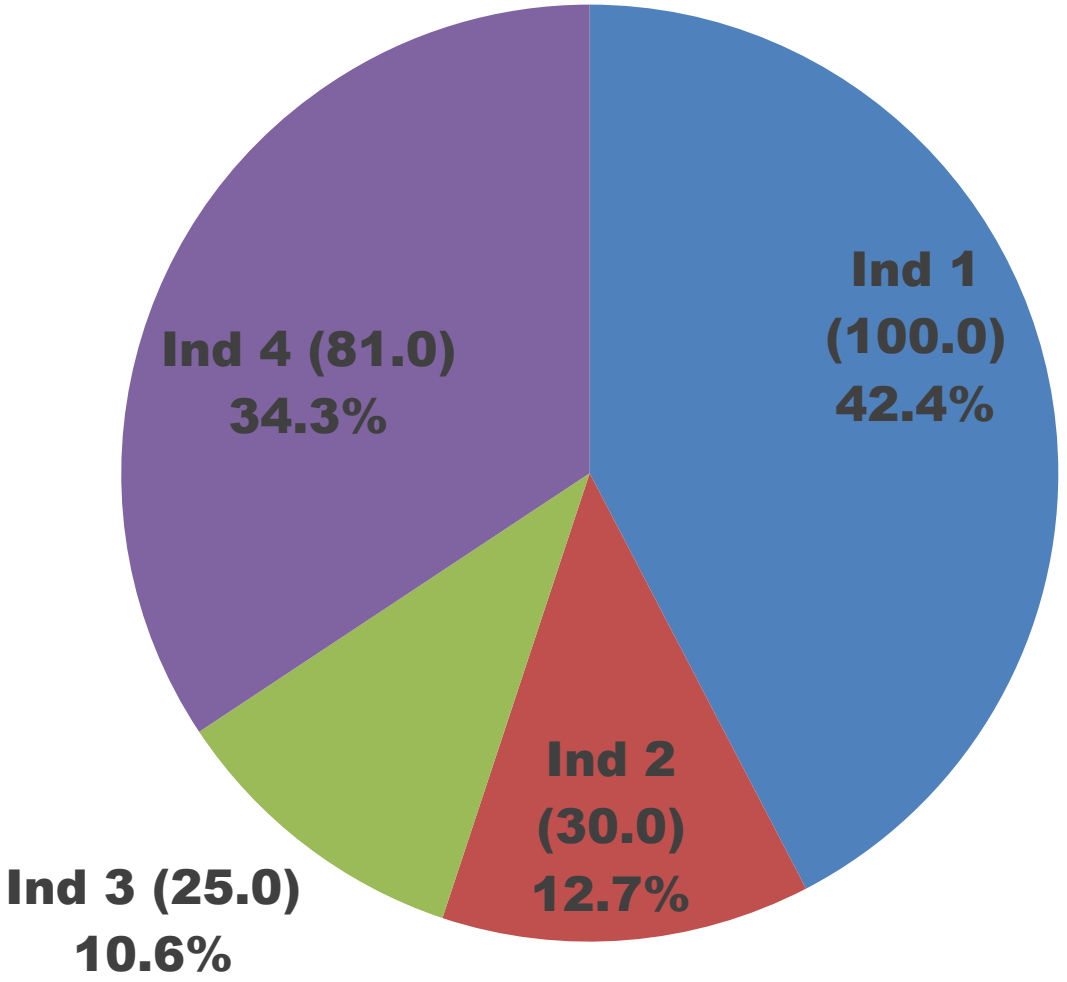
Fitness - remarks

- Let's assume that:
  - *f(x)* is the value of the optimized function
  - *x* is an individual that represents a solution
- Question, is it **always** true that:

  fitness(x) = f(x)
- **Not true!**
  - fitness != optimized function
  - Fitness can include penalty (wait and see)
  - Fitnes can proces the optimized function (e.g., revert it): *fitness(x) = 1/f(x)*

Genetic Algorithms
Roulette wheel

- Individual $(fitness(x_i))\ p_i(x_i)$
- Each individual has its chance
- Better individuals – higher chance
- Conclusion: roulette wheel **seems** to be a good selection technique



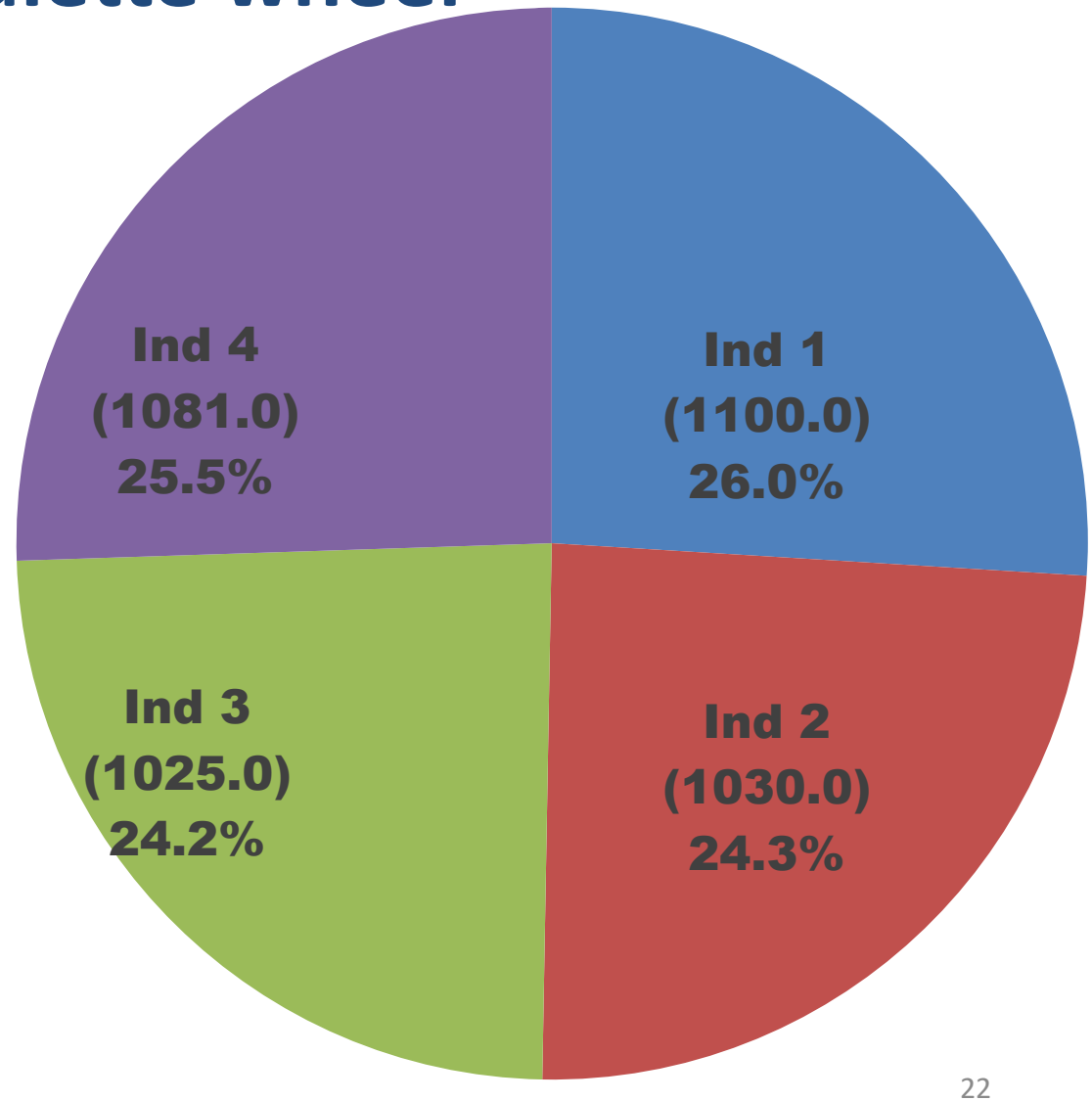
Wrocław University of Science and Technology

Genetic Algorithms
Roulette wheel

- Optimized function:

g(x) = f(x)+1000

- In general, g(x) and f(x) **are the same**
- But now, the probabilities are almost equal



Wrocław University of Science and Technology

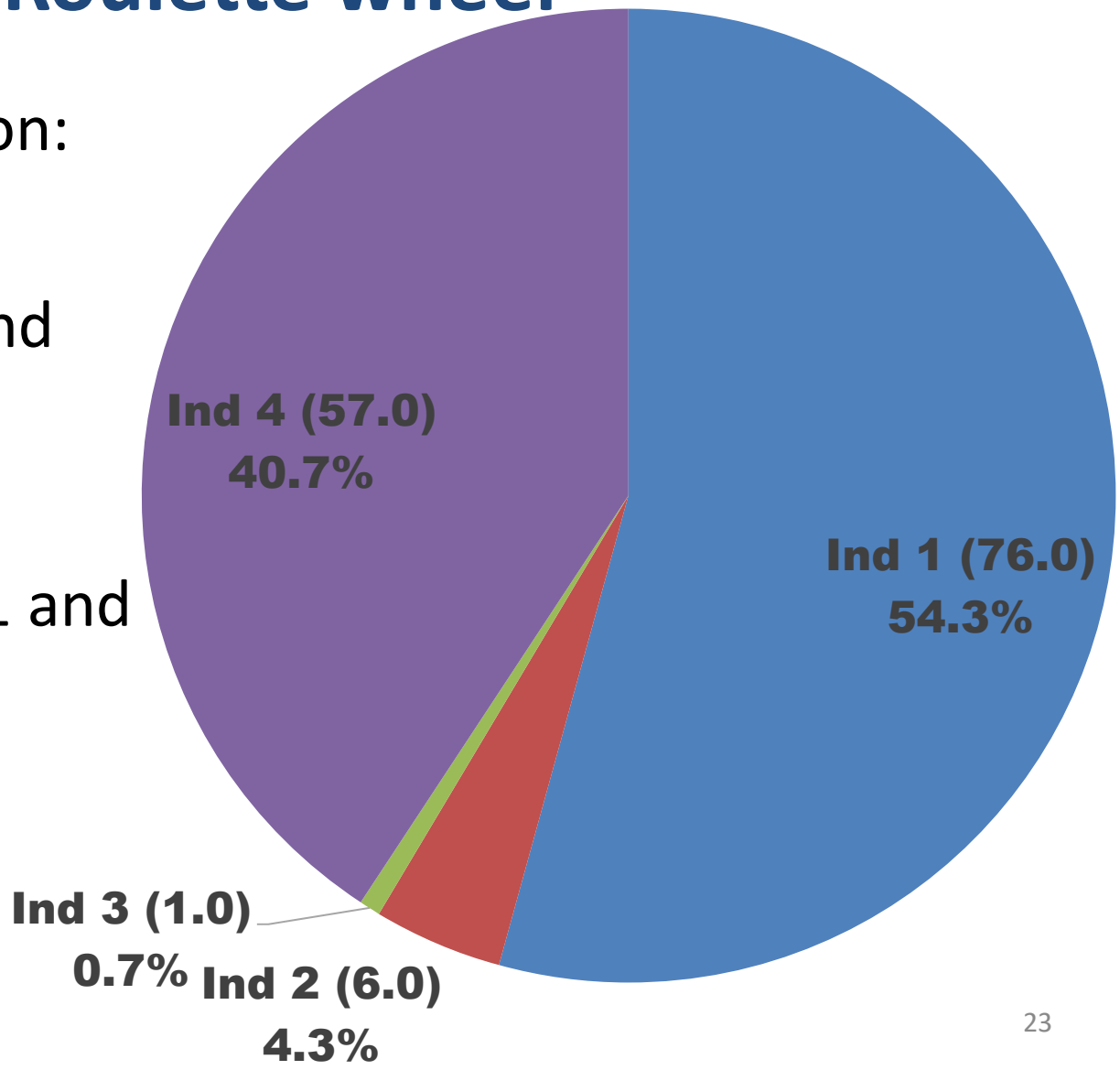
Genetic Algorithms
Roulette wheel

- Optimized function:

h(x) = f(x)-24

- In general, h(x) and f(x) **are the same**

- ..but individuals 1 and 4 **dominate**



Genetic Algorithms
Roulette wheel

- Slection based on roulette wheel
  - Highly dependent on the values of the optimized function
  - Makes controlling the convergence hard
  - Fitness must not be negative (cumbersome)

- **In general, state-of-the-art Gas do not use roulette wheel**

Genetic Algorithms

Tournament selection

- Tournament of size ***k***
  - Choose *k* individuals randomly (equal probability)
  - Among these *k* individuals choose the best individual as a winner
- Promotes best individuals (increasing convergence)

High *k* values

- Promotes diversity (a chance for lower-fitting individuals)

Low *k* values

Wrocław University
of Science and Technology

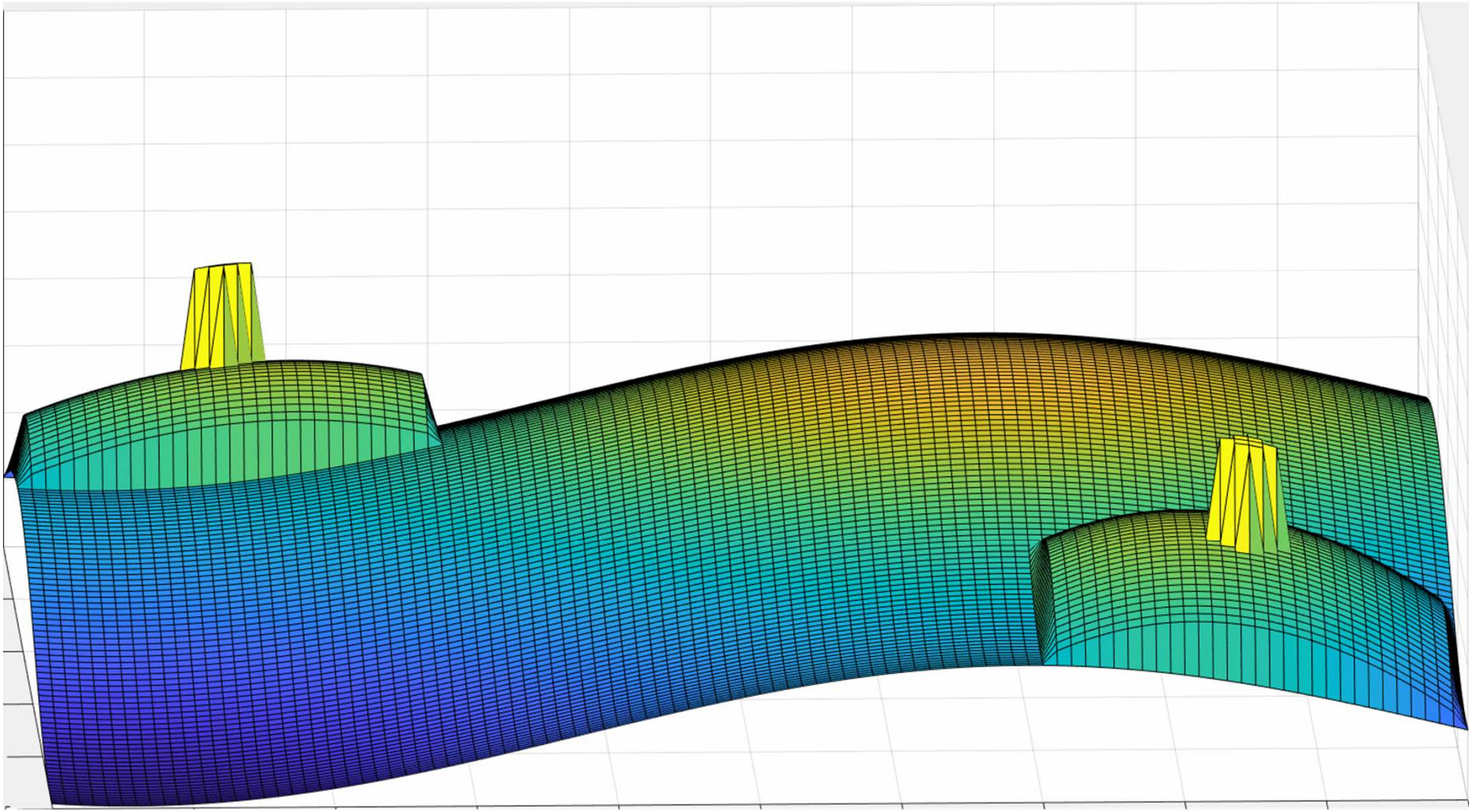
Genetic Algorithms

Tournament selection

- Population: 20 individuals
- Crossover probability: 0.4
- Mutation probability: 0.3

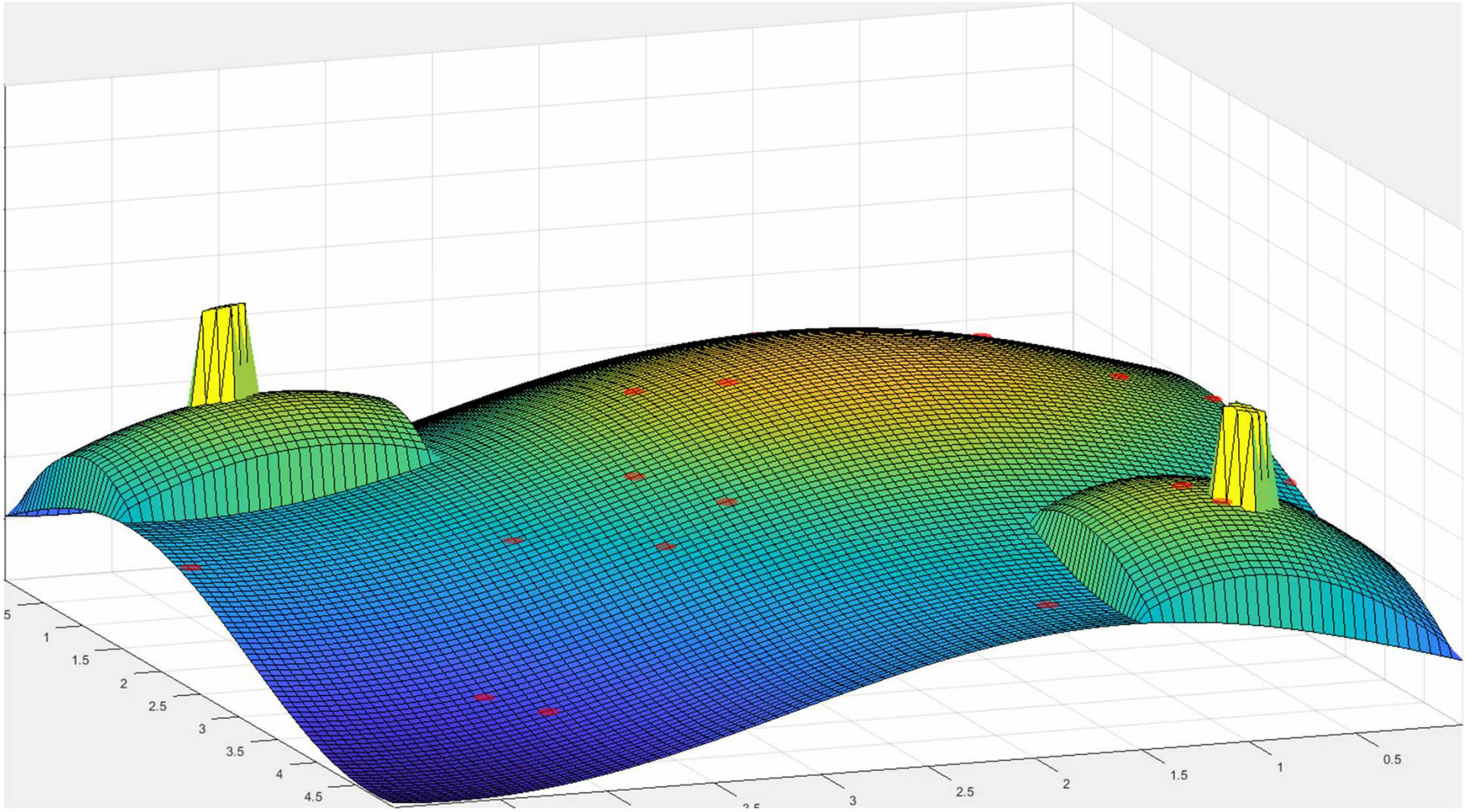
Genetic Algorithms

Tournament selection



Genetic Algorithms

Tournament selection *k=2* (normal convergence)



Genetic Algorithms

Tournament selection *k=2* (normal convergence)

- Optimum was found
- **Only** one of the two optima found

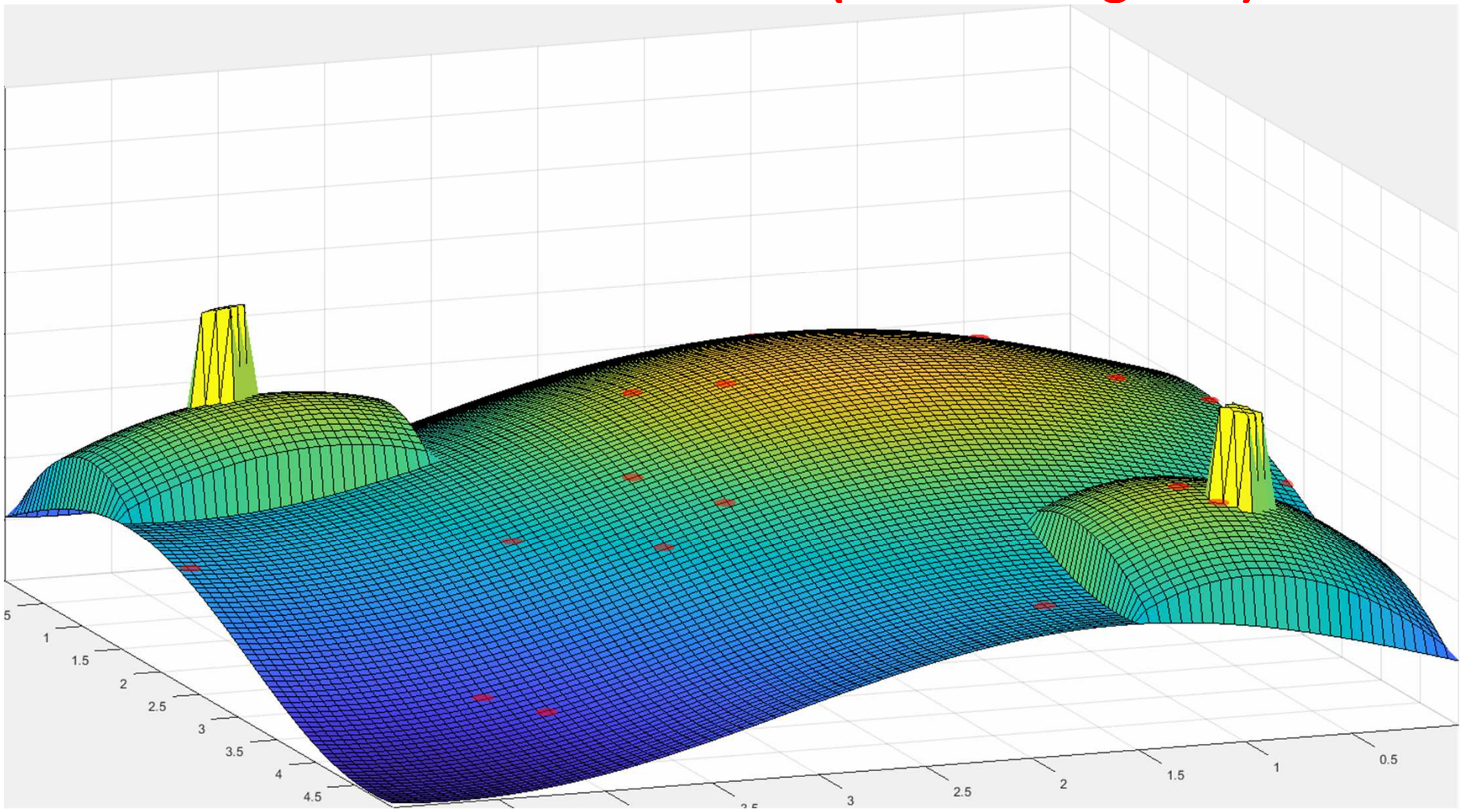
Genetic Algorithms

Tournament selection *k=2* (low convergence)

- Parent selection
  - In 90% of the cases – choose parent randomly, (fitness is ignored**)**
  - In 10% of the cases – tournament selection (k=2)

Genetic Algorithms

Tournament selection *k=2* (low convergence)



Wrocław University
of Science and Technology

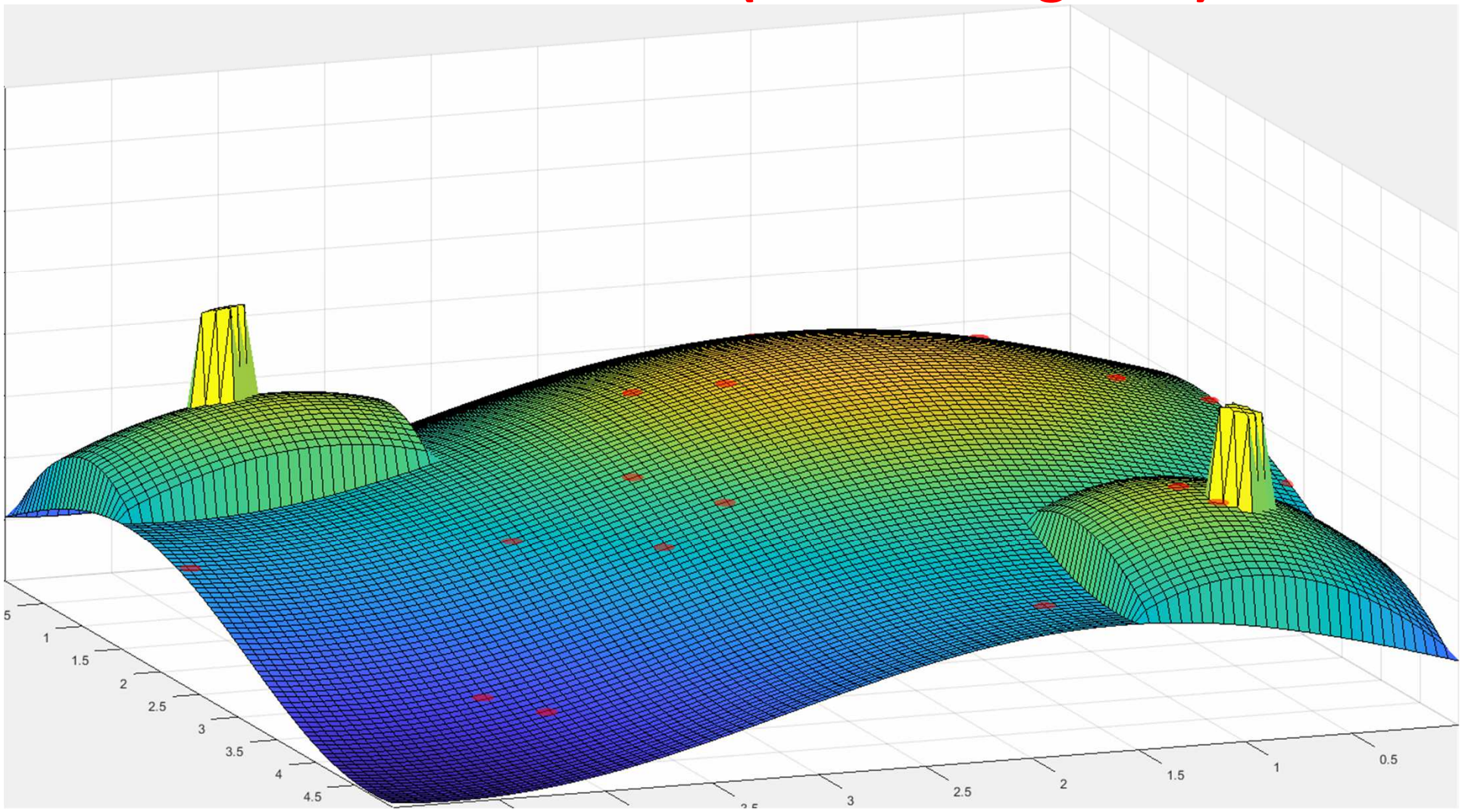
Genetic Algorithms

Tournament selection *k=2* (low convergence)

- Population runs aroung the space
- Convergence to random locations
- High-quality individuals are lost

Genetic Algorithms

Tournament *k=10* (PREconvergence)



Genetic Algorithms

Tournament *k=10* (PREconvergence)

- Optimum **NOT** found
- Individuals from the wide, but low hill dominated the rest of the population, **BEFORE** they had time to climb up

Genetic Algorithms

Convergence – how to handle that?

- **Convergence → one of the main issues in** optimization when GAs are used
- **Convergence** → directly connected to **population diversity**
- **Population diversity** → individuals located in difefrent regions of the search space
- **Population diversity** → one of the so-called **GA sweetspots**
- More about these issues → next lecture

Genetic Algorithms

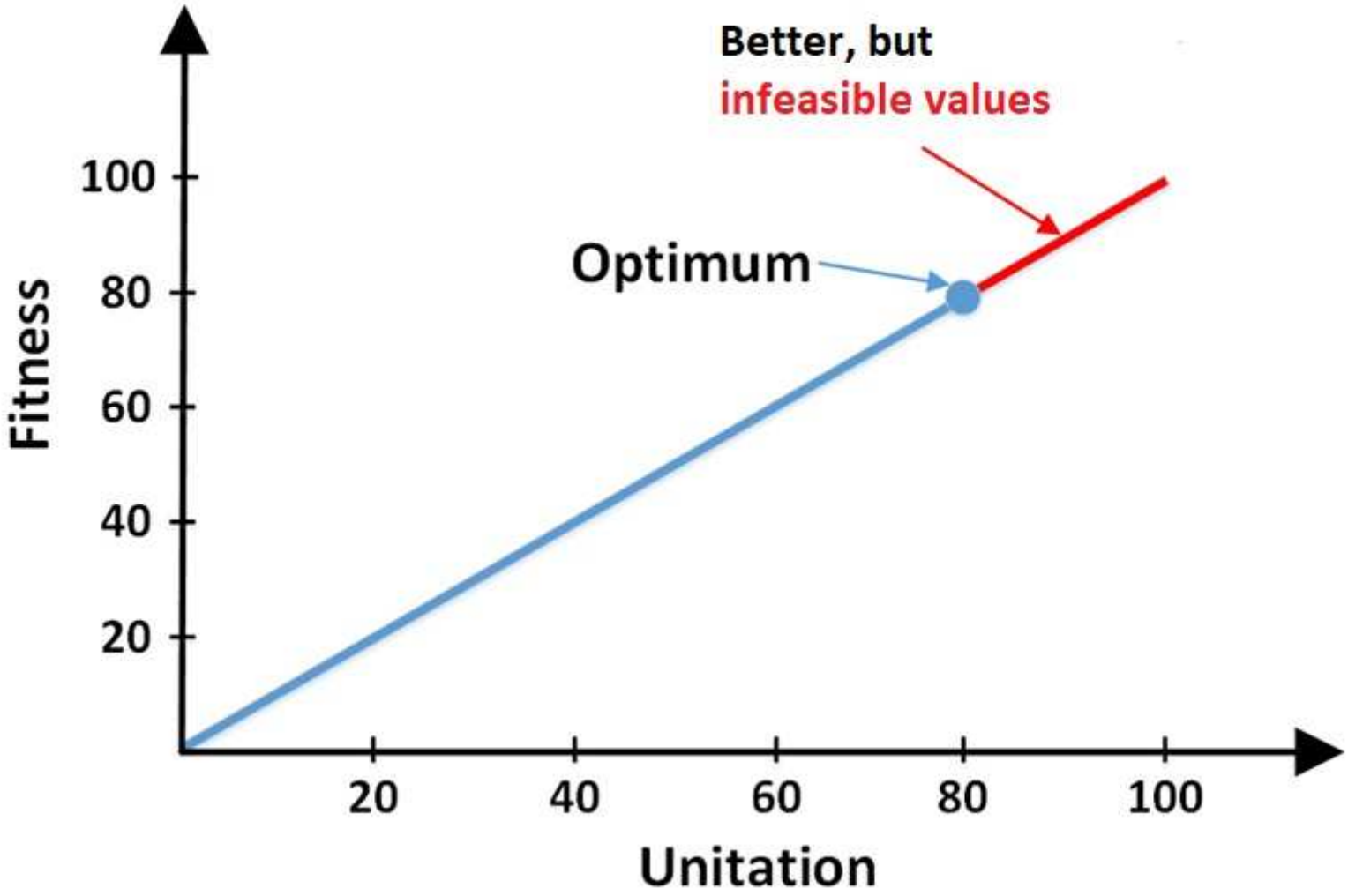
Constrained problems

- OneMax problem

$$OneMax(x) = u(x)$$

where $u(x)$ is *unitation* – the number of ‚1's in the genotype

- Let's consider the following *OneMax*:
  - Problem size: 100 genes
  - Constrained: if $u(x) > 80$, then solution is **infeasible**



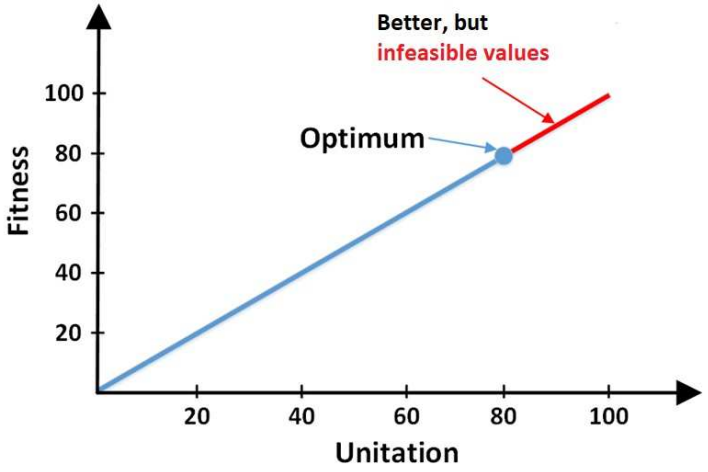
- What should be the fitness of infeasible individuals?
- Is individual with $u(x) = 81$ of low quality?

Genetic Algorithms

Constrained problems

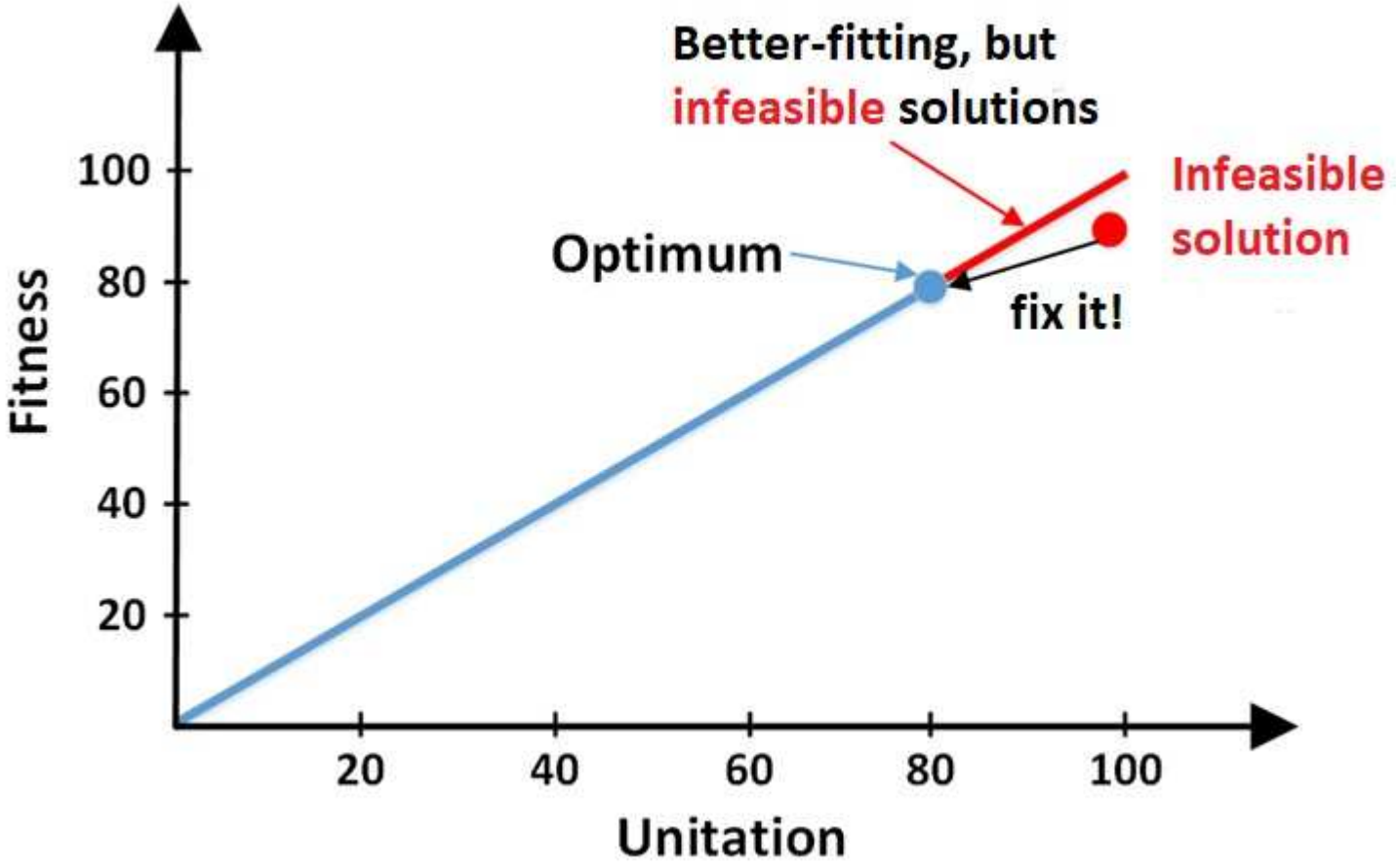
- Constraints
  - Directly defined
  - You can use this knowledge

    (e.g., define problem-deducat mechanisms)
- How to handle constraints?
  - „**fix**” solutions
  - use **penalty functions**



Genetic Algorithms
Constrained problems

- Fix solution



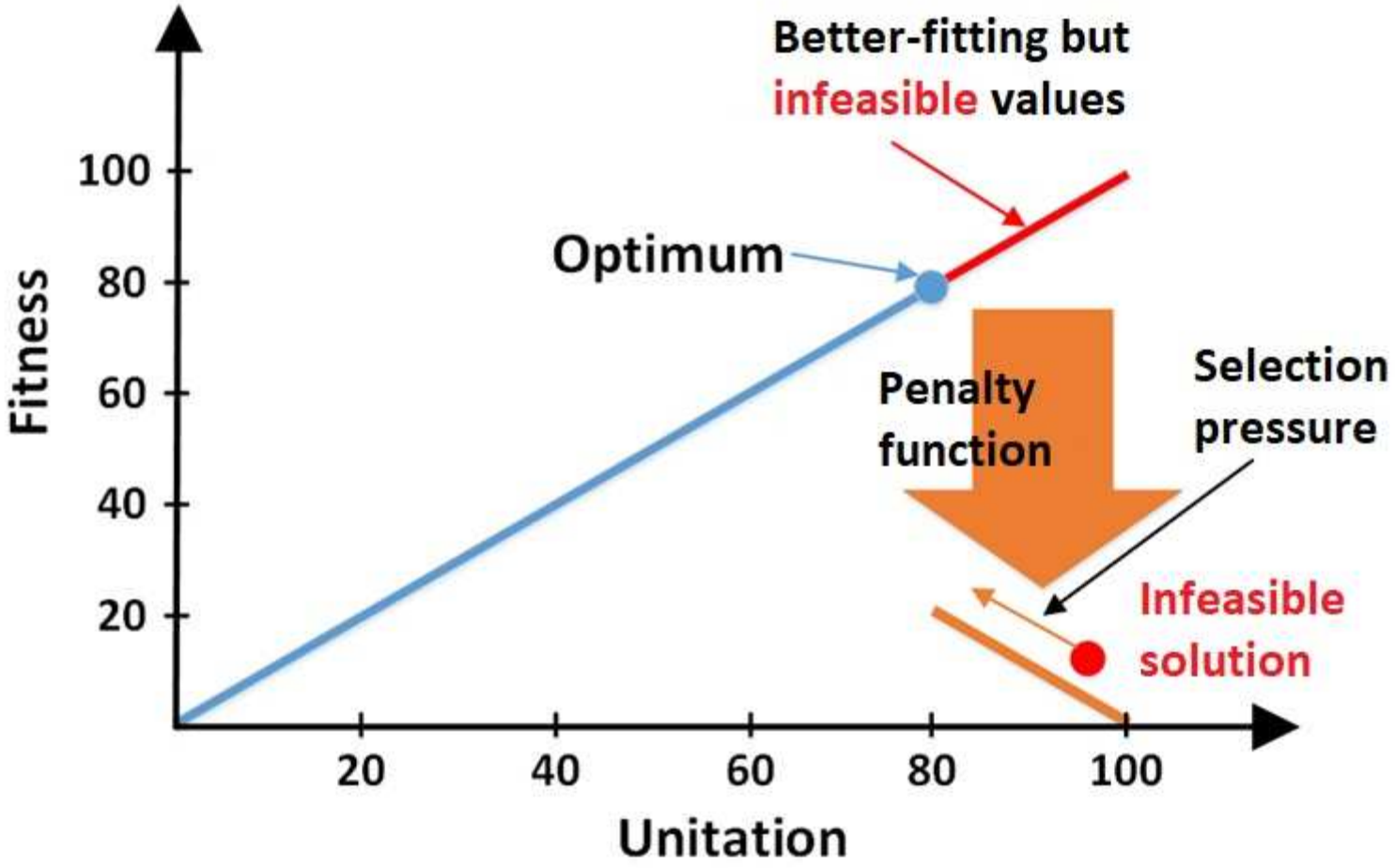
Genetic Algorithms
Constrained problems

- Solution fixing
  - Modify the **genotype**
    - Faster convergence (advantage)
    - Risk to stuck (disadvantage)
    - An example of the **Lamarck effect**
  - **Rate fixed genotype**, but leave the **original genotype unchanged**
    - Slower convergence (disadvantage)
    - Risk to stuck - reduced (advantage)
    - An example of the **Baldwin effect**
- **Baldwin and Lamarck effect – next lecture**

Genetic Algorithms

Constrained problems

- Penalty function



Genetic Algorithms
Constrained problems

- Penalty function
  - Modifies the shape of solution space
  - **Fitness != optimized function**
  - Usefull when:
    - Hard to define the fixing algorithm
    - Fixing algorithm is expensive

Wrocław University
of Science and Technology

Genetic Algorithms

Constrained problems

- Main techniques for handling infeasible solutions:
  - Solutionn-fixing algorithhms
  - Penalty function
- Other choices
  - Genetic operators that guarantee solution feasibility
  - Define problem as *multi-objective* → one of the next lectures
- Constraints
  - Frequent in real-world problems
  - Mechanisms for constraint handling → frequent optimizer adjustments

K. Deb and R. Datta "A fast and accurate solution of constrained optimization problems using a hybrid bi-objective and penalty function approach," In IEEE Congress on Evolutionary Computation (CEC), pp. 1-8, 2010.
M. W. Przewozniczek, R. Datta, K. Walkowiak and M. Komarnicki. Splitting the fitness and penalty factor for temporal diversity increase in practical problem solving. Expert Systems with Applications, vol. 145, pp.1-11, 2020

Genetic Algorithms
Computation load

- Optiimizer → must stop some day
  - Good-enough solution found **(Attention! It may not happen!)**
  - Iteration number
  - Computation time
  - Fitness function evaluations
- **You want to know which optimizers works better?**
  - Computation load measurement → **you must have it**
  - **You know when to stop computation**
  - **You know what was the cost of finding the best solution**

Genetic Algorithms
Computation load

- Iteration number
  - Is it **reliable**?
  - It ignores the iteration cost
- Example:
  - two GAs
  - The same prooblem
  - First GA –1000 individuals
  - Second GA – 20 individuals
  - **Is the cost of the single iteration the same?**

Genetic Algorithms
Computation load

- Example:
- GA1:
  - **Popuulation: 1000**
  - Crossover: 0.6
  - Mutation: 0.1
- GA1:
  - **Popuulation: 1000**
  - Crossover: 0.2
  - Mutation: 0.05
- **Is iteration number reliable in this case?**

Genetic Algorithms
Computation load

- The example from the previous slide, answer: **no, it's not**
- Why?
  - In each iteration we create a new population
  - First, we do the selection
  - The, we perform crossover with P_{cross} probability (here: 0.6 or 0.2)
  - Conclusions?
    - GA1: after crossover **40% of indiduals are the copies of their parents**
    - GA2: after crossover **80% of indiduals are the copies of their parents**

Genetic Algorithms
Computation load

- Why iteration number is not reliable? - continuation
  - After crossover:
    - GA1: **40% of indiduals are the copies of their parents**
    - GA2: **80% of indiduals are the copies of their parents**
  - After mutation (GA1: 0.1; GA2: 0.05)
    - GA1: **36% of indiduals are the copies of their parents**
    - GA2: **76% of indiduals are the copies of their parents**
  - **Do we have to re-evaluate the copy of the parent?**

Genetic Algorithms
Computation load

- literation number as a stop condition – when to use?
  - Compare/check optimizer's behaviour (e.g., the modification influence on the optimizer's run)
  - Run analysis of a single optimizer
- Optimization effectiveness comparison
  - Results quality comparison
  - **In general, iteration number is not reliable**
  - **Without a convincing arguments:**
    - **Severe mistake**
    - **Results - unreliable**

Genetic Algorithms
Computation load

- fitness function evaluation number (FFE)
  - Some time other (similar) names are used
  - Frequent measure in the „well-published” research

Genetic Algorithms
Computation load

- FFE – pros
  - Hardware independent
  - Easy to parrarelize the experiments
  - Easy to compare
    - You have the results from the literaturę?
    - You can copy the general results of the other optimizer
    - **„*Copy-Paste research*" – a good practice?**
      - **You can not extend the research (other problems, other FFE-based stop condition)**
      - **You can not check the copied results**
      - **Leading journals and conferences – „I can not extend the research" = paper rejected**

Genetic Algorithms
Computation load

- FFE – myths and facts
  - Independent of the source code quality?

    Not true. Example: you do not have to re-evaluate the copies of the parents
  - Most reliable?
    - **What if the computation load is mostly dependent on something else than FFE?**
    - Example: *Bayesian Optimization Algorithm* (BOA) → *Esimation of Distribution Algorithm* (EDA) → highest cost goes for the model

H. Kwasnicka and M. Przewozniczek, "Multi Population Pattern Searching Algorithm: a new evolutionary method based on the idea of messy Genetic Algorithm" in IEEE Transactions on evolutionary computation, vol. 15, no. 5, pp. 715-734, 2011

Genetic Algorithms
Computation load

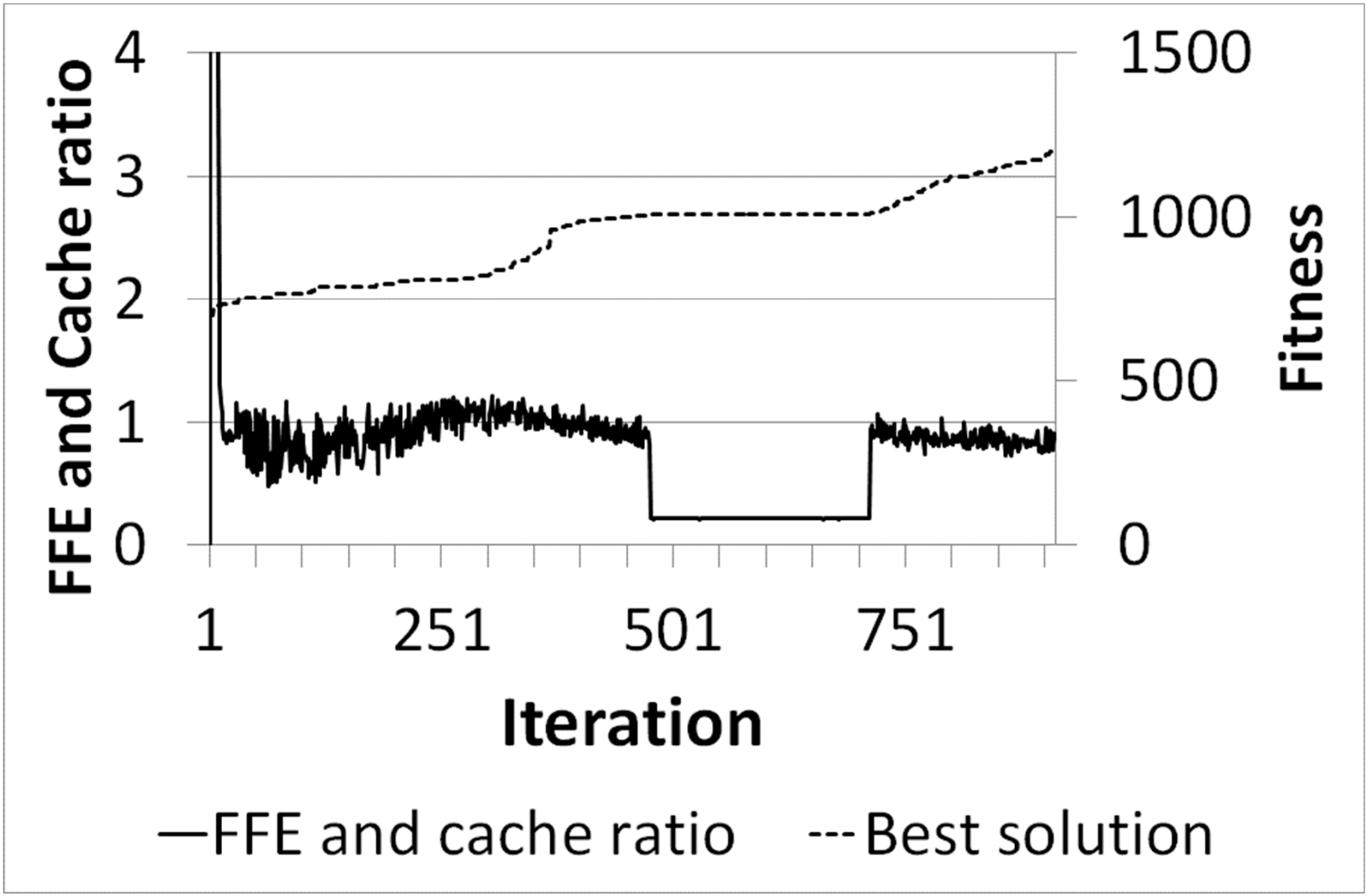
- FFE minimization
  - fitness caching
    - Global list of „already rated” solutions with the computed fitness
    - Evaluate genotype:
      - Check if the genotype is on the list
        - » Yes → return stored fitness
        - » No →evaluate and store the pair (genotype, fitness)
    - List search
      - High searching cost
      - High memory usage

M. W. Przewozniczek and M. M. Komarnicki, “The influence of fitness caching on modern evolutionary methods and fair computation load measurement,” in Proceedings of the Genetic and Evolutionary Computation Conference Companion, ser. GECCO ’18. ACM, 2018, pp. 241–242.

Genetic Algorithms
Computation load

- fitness caching – continued
  - Effects
    - Effective FFE minimization **(but useful in practice?)**
    - **If method is stuck → it checks the same genotypes → FFE not rises → computation will never end**
  - Global list → high memory and CPU cost
  - Alternative → population is the buffer
    - similar effect
    - Low costs
- ATTENTION: if optimizers buffers FFE, the **FFE may not be a reliable computation load measure anymore**

Genetic Algorithms
Computation load



- FFE caching – optimum found

Genetic Algorithms
Computation load

- FFE minimization
  - *FFE caching (discussed)*
  - Surrogate models
    - Model „learns” the optimized function
    - Fitness → computed using the model
    - Disadvantage → model may be inaccurate
    - Regular fitness is expensive (minutes/hours of computation)? → **use surrogates!**

D.R. Jones, “A taxonomy of global optimization methods based on response surfaces” in Journal of Global Optimization, vol. 21, pp.345–383, 2001.

Genetic Algorithms
Computation load

- FFE **cost** (not number) minimization
  - Some problems:
    - Solution evaluation – the sum of many elements
    - Modified solution
      - Only **some** elements are modified
      - Re-evaluate **only the modified** elements
      - The evaluation of the rest of the elements – use the stored values

A. Bouter, T. Alderliesten, A. Bel, C. Witteveen, and P. A. N. Bosman. "Large-scale parallelization of partial evaluations in evolutionary algorithms for real-world problems." In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '18), pp. 1199–1206, 2018.

Genetic Algorithms
Computation load

- FFE **cost** (not number) minimization - continued
  - Other problems:
    - Model necessary (e.g., of a computer network) for solution evaluation
    - Modified individual
      - Copy the original model
      - Modify the original model
      - Re-evaluate the modified modelu

M. W. Przewozniczek and M. M. Komarnicki. The influence of fitness caching on modern evolutionary methods and fair computation load measurement. In Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO '18, page 241-242, 2018.

Genetic Algorithms
Computation load

- FFE as a computation load measure
  - Fitness caching
  - Surrogate models
  - FFE cost minimization
  - **FFE may not be reliable anymore**
- **Always analyze if your stop conditio is reliable**
- **Unreliable stop condition → unreliable experiments**

Genetic Algorithms
Computation load

- **When to use FFE?**

Always when it is justified:

$$T_{FFE} \gg T_{Other}$$

Where:

T_{FFE} - summarized cost of all FFE

T_{Other} - summarized cost of all operations **OTHER** than FFE computation

T_{FFE} - linearly (or close to linearly) dependent on FFE

Genetic Algorithms
Computation load

- Computation time
  - Disadvantages:
    - The same computer → the same seed → siginificantly different Times possible
    - Reliable comparison → only on the same computer
  - **Advantages – true and complete computation cost**
  - How to?
    - Share all the possible source code parts (problem implementation, operators, etc.)
    - Run experiments on „clean” system
    - Maintain the same number of computation processes

Genetic Algorithms
Computation load

- Choose wisely
- Analyze if your computation load measure is reliable
- Wrong choice → unreliable results
- Unreliable results → unreliable conclusions **(a death kiss to your research)**

Genetic Algorithm
Is it an algorithm?

- Algorithm
  - A serie of determined instructions that…
  - …solve the problem
- Algorithms must:
  - Be *correct* → return *correct* solution
  - Finite → they must end some day…
- **Is it true for GAs?**
- Method → does not have to follow the above demands
- Therefore, here → **GAs are methods** (or optimizers)
- **Attention: many respected scientists claim that GAs so-called randomized algorithms**
  - Therefore, GAs are frequently denoted as *algorithms*
  - Naming convention → irrelevant → it is about term definiton

BUT WHY DOES it WORK???

SCHEMA THEOREM

- 50-years old
- Unrealistic assumptions
- More like science fiction…

BUT STILL

- It is the only gate to understan GAs
- You will get all the necessary intuitions
- These intuitions are still up-to-date

Optimization – what is it?

The knapsack problem

- *n* items
- Each item has two features:
  - Weight
  - Value
- Objective: pack the most valuable item to your knapsack
- Constraint: the summarized weight of packed items can not be larger than *W*

Optimization – what is it?

The knapsack problem

- How to encode a solution?
- *n*-bit vector
  - Each bit refers to the single decision:
    - I pack this item (1)
    - I do not pack this item (0)
- Solution example (5 items):
  - 01100
  - I pack items 2 and 3

Genetic Algorithms

Reminder

- Original objective: evolution simulation
- **Current objective**: optimization
- Basic terms
  - Population of individuals
  - Individual = encoded solution
  - Fitness → related to solution quality, i.e., the optimized function (not necessarily the same)
  - Selection → Higher fitness → higher Chance for being a parent

Genetic Algorithms
Reminder

- Basic terms (continued)
  - Crossover
    - Two parent
    - Two offspring
      - Parent copies or…
      - Mixed parents genotypes
  - Mutation → random modification of a (usually small) part of the genotype

Genetic Algorithms

Crossover and mutation - examples

- Fitness → function value
- Solution → individual → genotype

  2 real numbers from the interval [0; 5]
- **Corssover**:

1st offspring: 1st gene of ***prent 1*** + 2nd gene of ***parent 2***

2nd offspring: 1st gene of ***prent 2*** + 2nd gene of ***parent 1***

Mum: <2.5; 2.8> Dad: <1.2; 4.8>

Doughter: <2.5; 4.8> Son: <1.2; 2.8>

- Mutation:

  add random value from the interval [-0.15; 0.15]

Genetic Algorithms

Crossover and mutation - examples

- Fitness → summarized values of items in the knapsack
- Problem instance → 7 items
- Solution → individual → genotype →

  7 binary values
- **Crossover**:

Mum: 1110011 Dad: 0101010

Doughter: 1111010 Son: 0100011

- Mutation: flip gene value with probability = 0.05

Doughter: 1111010 Son: 0***0***001***1***11

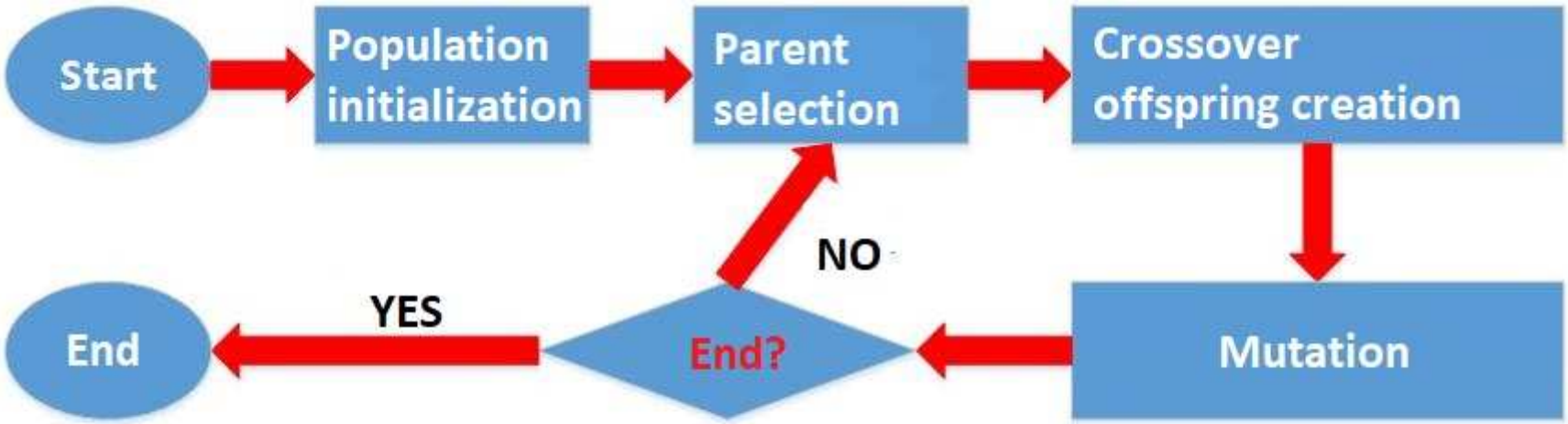
Genetic Algorithms

Reminder

- **Until the end of this lecture → binary coding**
- Why?
  - Theory works → usually use it
  - High benchmark problem number
  - How to do something important?
    - Propose an optimizer/a mechanism on the high generality level
    - Show that it useful for hard benchmarks with different features
    - Show that it works for real-world problems too
- Some real-world problems → also use binary coding

Genetic Algorithms

Reminder



Genetic Algorithms
Roulette wheel

- Chance for selection – proportional to fitness

$$p_i(x_i) = \frac{fitness(x_i)}{\sum_{j=1}^{size} fitness(x_j)}$$

where:

- x_i – *i*th individual
- $p_i(x_i)$ – the probability of choosing the *i*th individual
- $fitness(x_i)$ - fitness of the *i*th individual
- size – population size

Test problem

OneMax

$$f(\vec{x}) = u(\vec{x})$$

where:

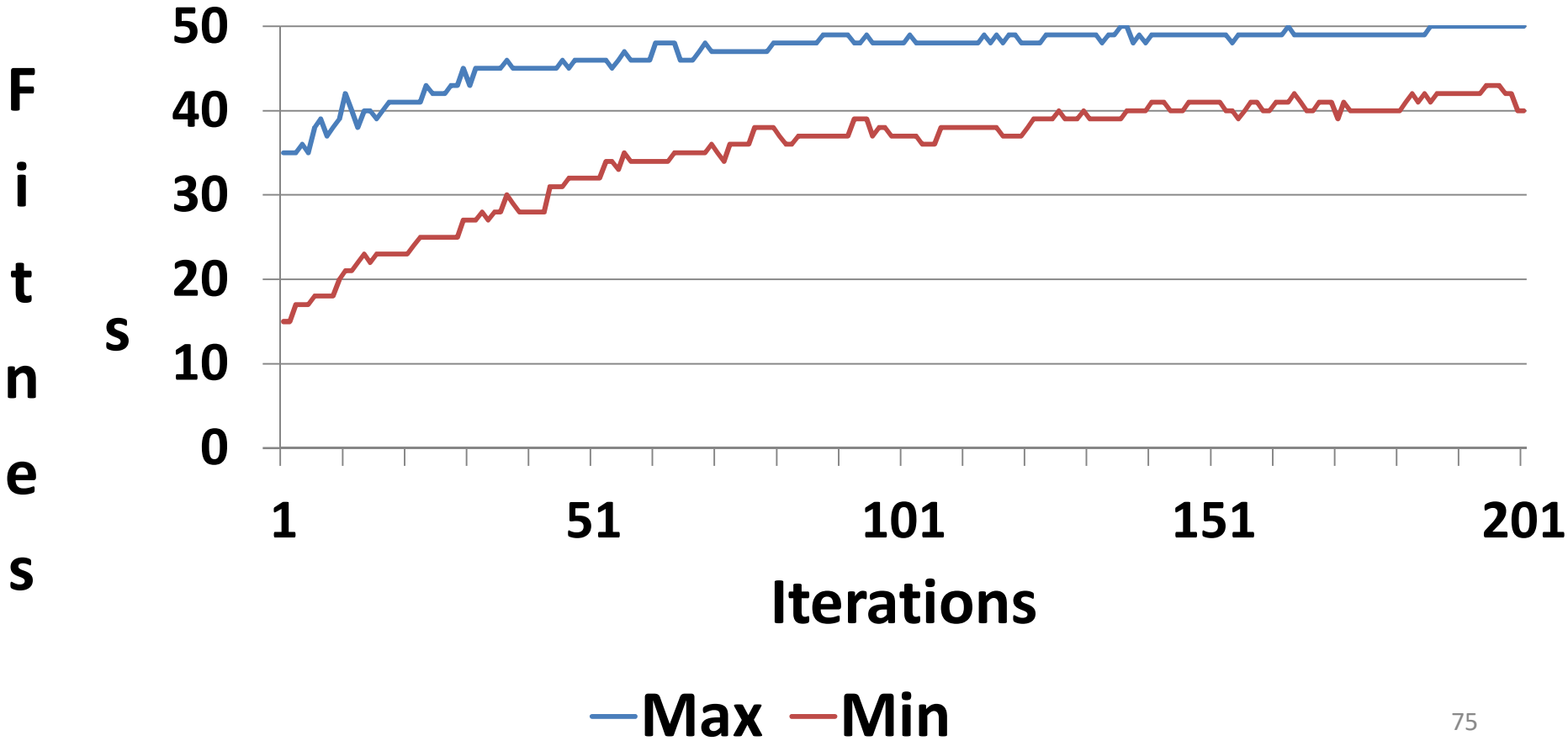
- $\vec{x}$ - binary vector (solution)
- $u(\vec{x})$ - unitation (the number of 1s in the genotype)
- Examples:

$u(1101) = 3$

$u(0001) = 1$

GA – simple test

Pop: 400; Crossover: 30%; Mutation: 0.01%



Test problem

deceptive functions concatenations

$$f_{order-k}(\vec{x}) = \begin{cases} u(\vec{x}) & if \quad u(\vec{x}) = k \\ k - u(\vec{x}) - 1 & if \quad u(\vec{x}) < k \end{cases}$$

where:

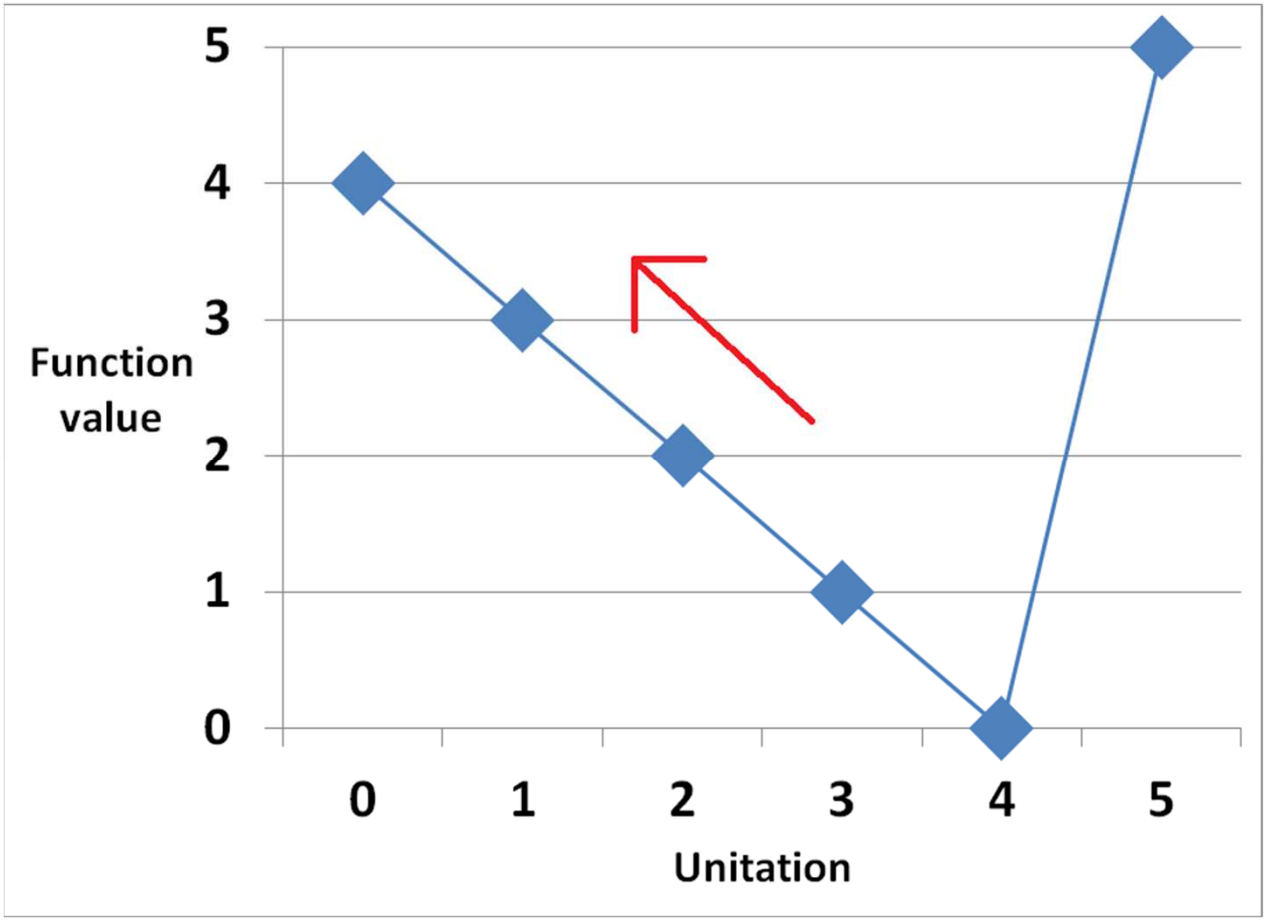
- $\vec{x}$ - binary vector (solution)
- $u(\vec{x})$ - unitation
- k – the order (size) of the deceptive function

Test problem

deceptive functions concatenations

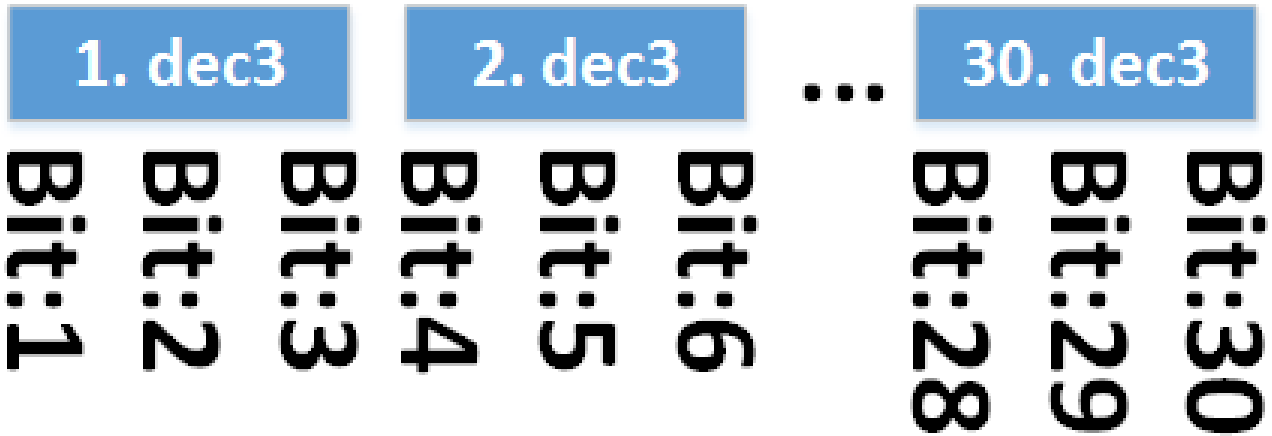
$f_{order-5}(x)$:

Unitation	Wartość
0	4
1	3
2	2
3	1
4	0
5	5



GA – a harder test

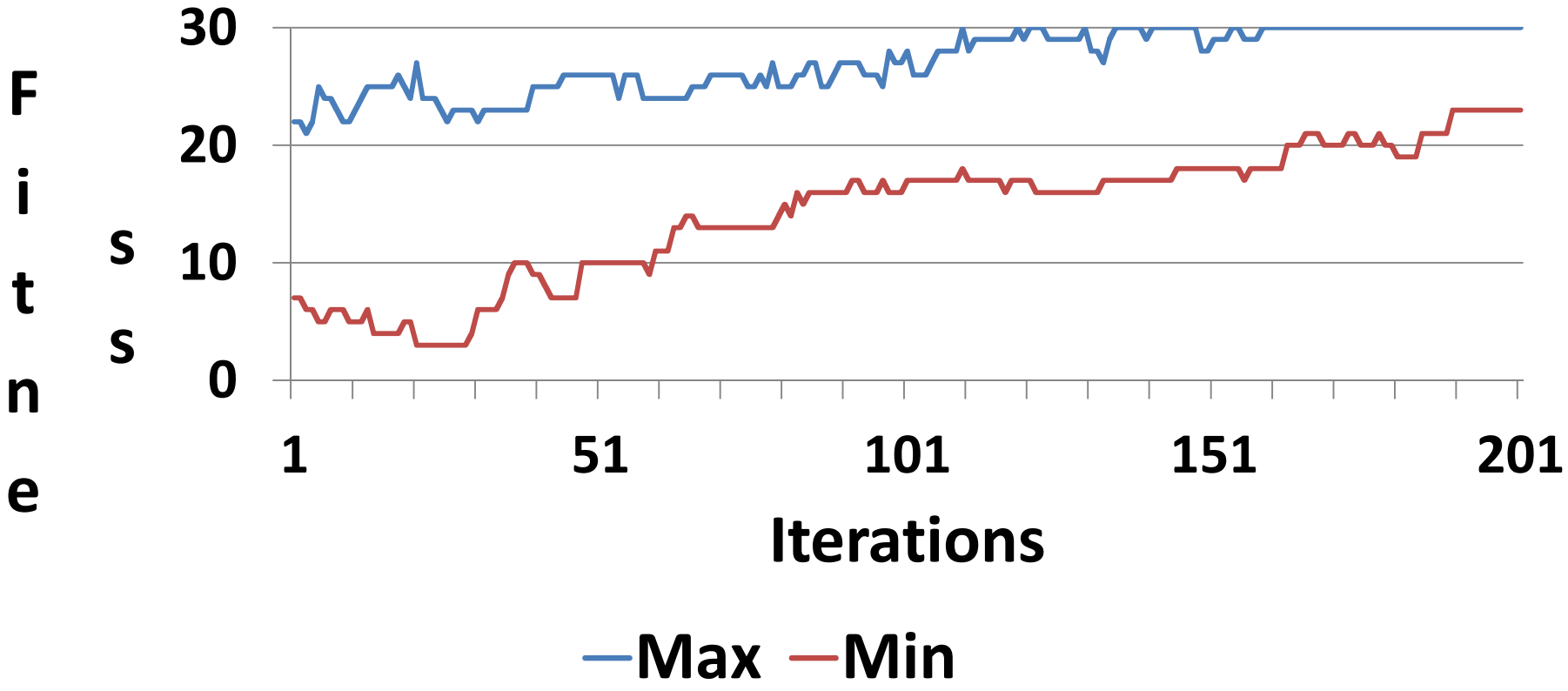
Concatenation of 30 funkcji deceptive functions with *k=3*:



Can GA solve it?

GA – a harder test

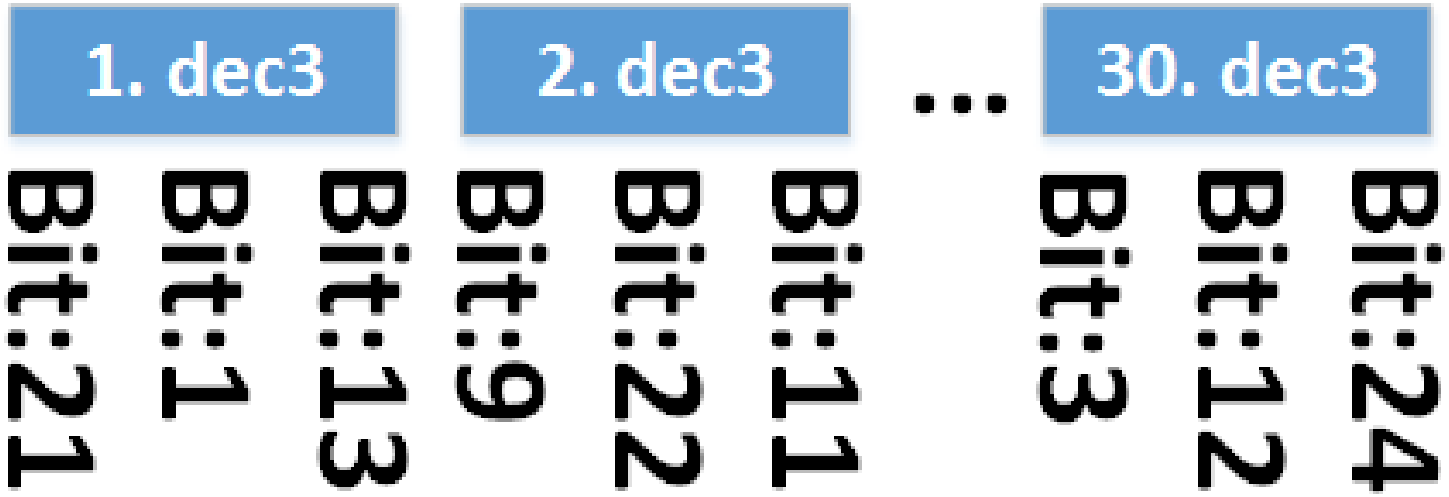
YES! GA solved it!



Pop: 400; Crossover: 30%; Mutation: 0.01%

GA – a harder test

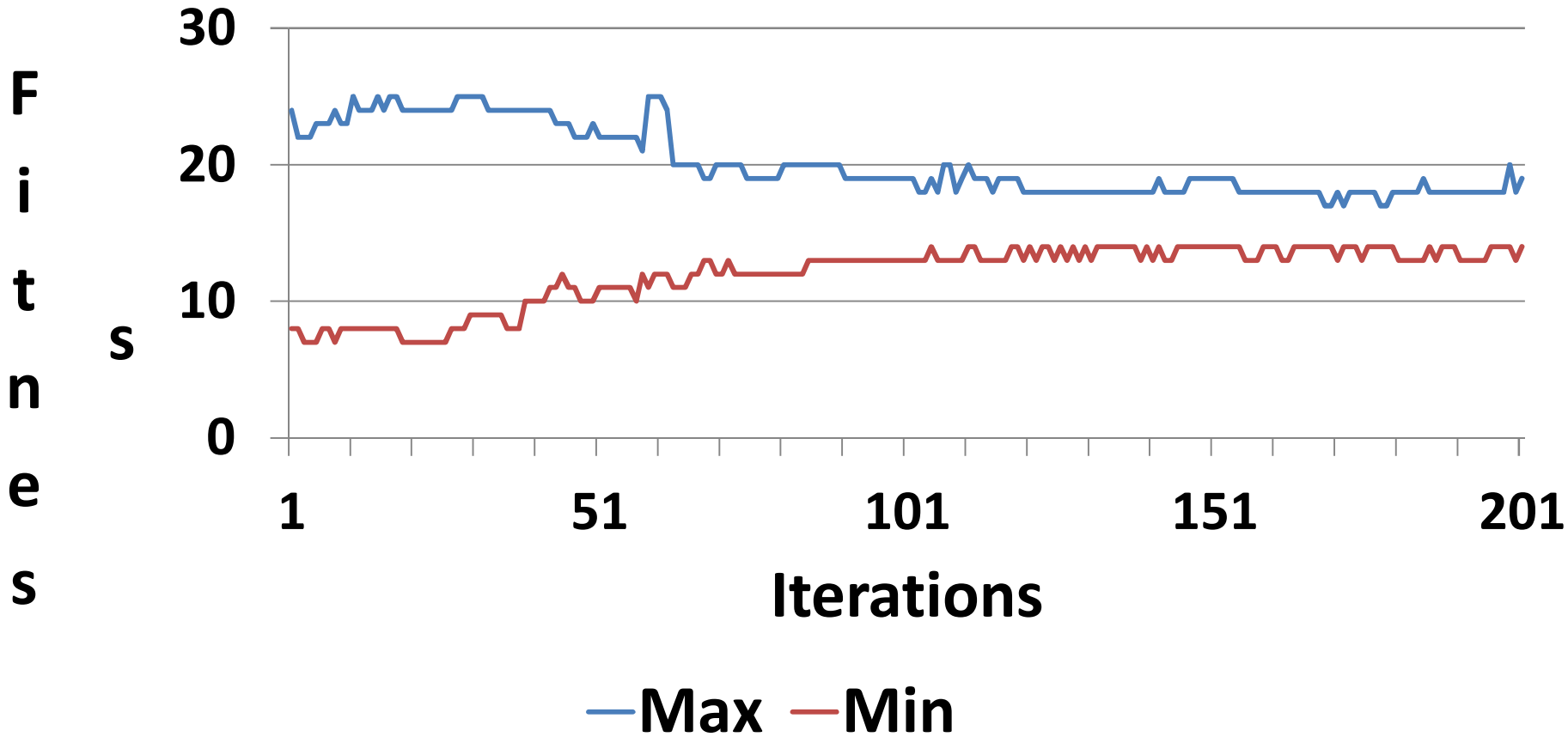
Let’s shuffle the bits:



How is it going to be this time?

GA – trudniejszy test

Ddoesn't work…



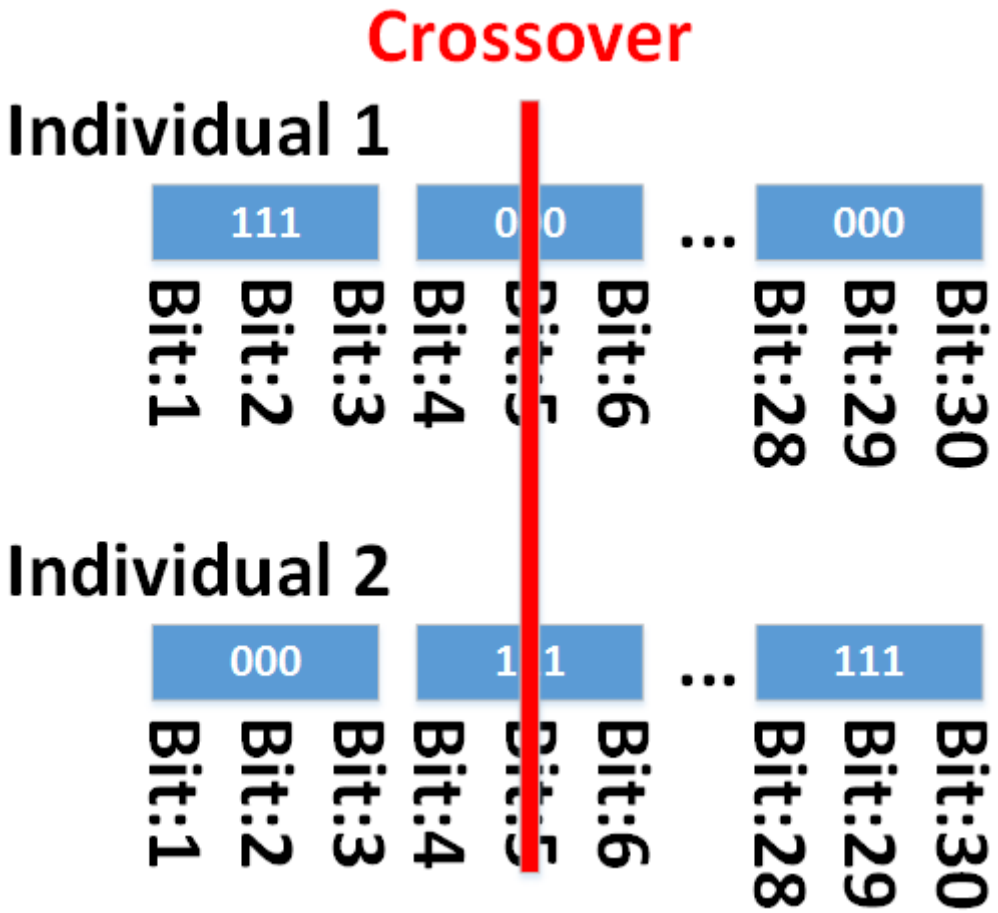
Pop: 400; Crossover: 30%; Mutation: 0.01%

Pytania po testach

- **Random gene order – why GA did not solve it?**
- Deceptive funciton concatenations
  - Key feature?
  - **Blocks of dependent genes!**
  - How to solve it?
  - **Exchange blocks!**
- Crossover – how does it **actually** work?

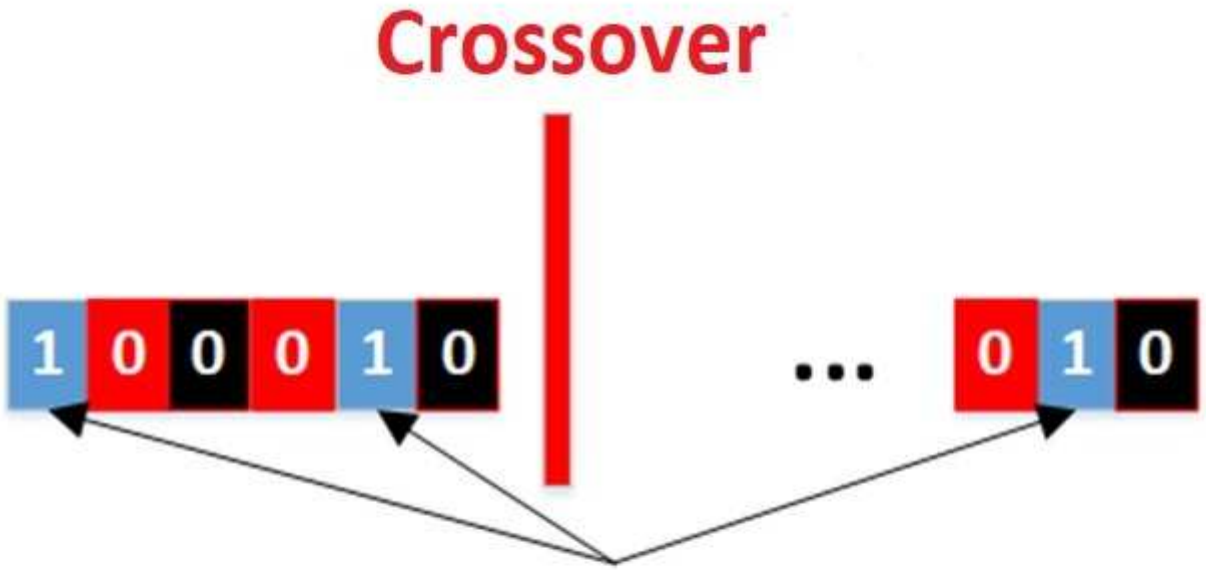
Single point crossover and the blocks

- **Each block converges to**
  - **000**
  - **111**
- **Crossover – how many blocks do we exchange?**
  - **All**
  - **All-1**



Single point crossover and the blocks

- And **AFTER** gene shuffling?



Some blocks will be always cut into pieces

Wrocław University of Science and Technology

Crossover and the blocks

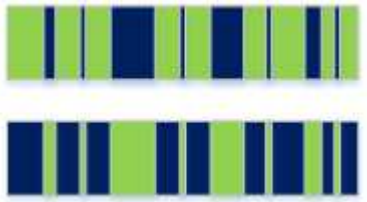
- Crossover can influence the way GA works
- What are the main choices?

1-point

Doesn't work

2-point

Almost the same

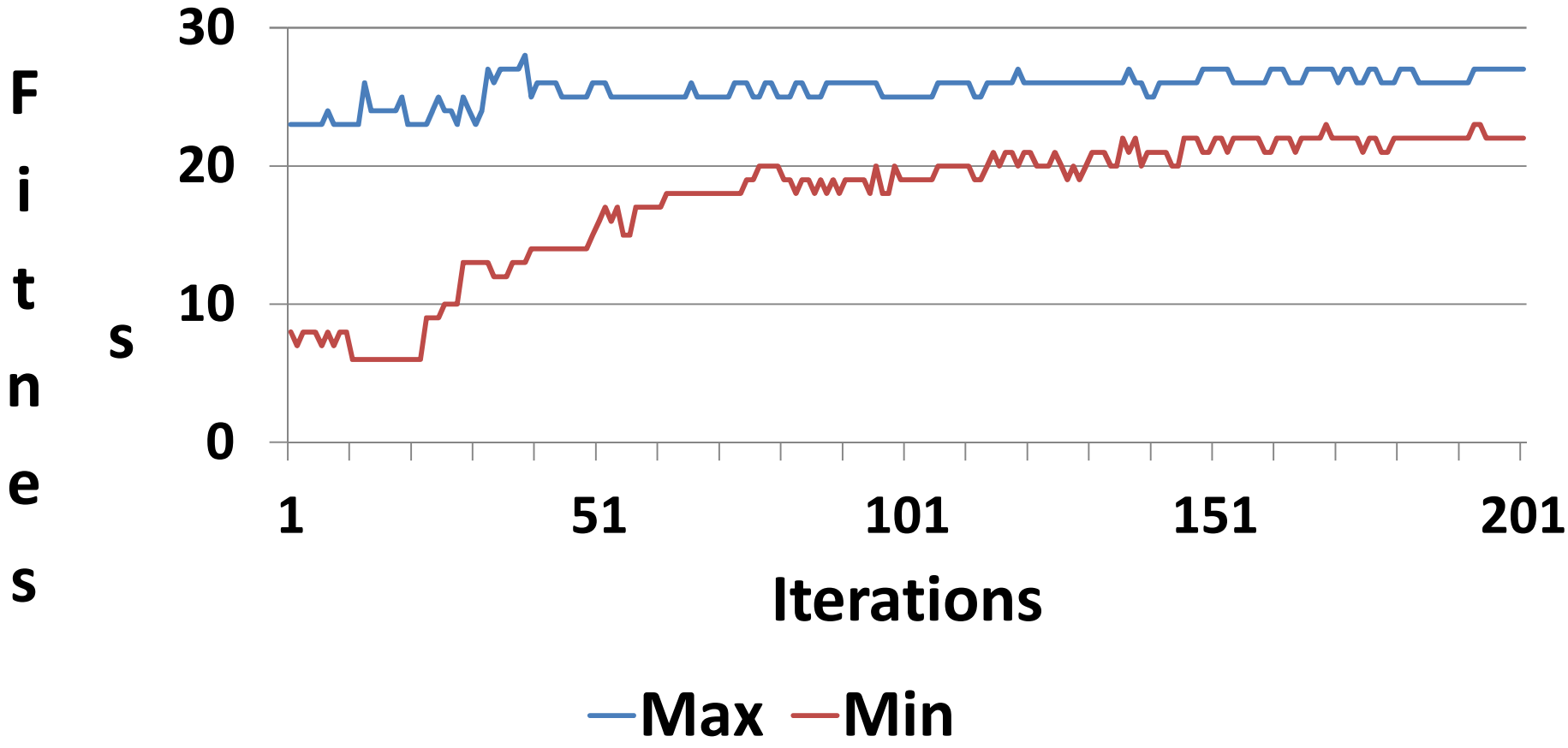


Uniform

Significantly different – let's try it!

GA with uniform crossover

Better, but GA did not find optimum…



Pop: 400; Crossover: 30%; Mutation: 0.01%

Wrocław University
of Science and Technology

Schema theorem

Why GA works?

Why GA DOES NOT work?

Schema theorem

- Binary coding
- Genotype example (length: 10):

  11100 10100
- Schema
  - „*” – goes for any value (wildcart)
  - Schema example (length: 10):
    - *1100 0101*
    - Genotypes fitting to the schema:
      - **0**1100 0101**0**
      - **0**1100 0101**1**
      - **1**1100 0101**0**
      - **1**1100 0101**1**

Schema theorem

- Genotypes fitting the schema
  - 00101 00101 – only 1 genotype fits
  - ***** ***** – 2^{10} genotypes fit
  - Every schema – 2^{r} genotypes fit to it (*r* – the number of wildcarts *)
  - Every genotype – fits to 2^{n} schema (*n* –gene number)
  - For genotype: 01000 11011

- 01000 11011
- *1000 11011
- 0*000 11011
- …

- 01000 1101*
- **000 11011
- *1*00 11011
- …

- 01000 110**
- ***00 11011
- …
- ***** *****

Schema theorem

- Schema and the population; For *n* genes:
  - 3^n of available schema
  - For populationof size *pop*, the number of schema in the population equals:
    - 2^n (all individuals the same)
    - Up to $pop \cdot 2^n$ **(that it why population diversity is important)**

Schema theorem

- *o(S)* – the **order** of schema *S*
  - Number of specified positions
  - Examples:
    - S_1 = (***11 0*0*0)
    - S_2 = (***** 11***)
    - S_3 = (01001 0**11)
  - $o(S_1) = 5$
  - $o(S_2) = 2$
  - $o(S_3) = 8$

Schema theorem

- *δ(S)* – **defining length** of schema *S*
  - The distance between the first and the last specified position
  - Examples:
    - S_1 = (***11 0*0*0)
    - S_2 = (***** 11***)
    - S_3 = (01001 0**11)
  - $\delta(S_1) = 10 - 4 = 6$
  - $\delta(S_2) = 7 - 6 = 1$
  - $\delta(S_3) = 10 - 1 = 9$

Schema theorem

- GA the general pseudocode:

$t \leftarrow t + 1$

Choose *P(t)* z *P(t-1)*

Cross and mutate *P(t)*

Evaluate *P(t)*

- GA key mechanisms:
  - Selection
  - Crossover
  - Mutation

Schema theorem

- $\xi(S,t)$ – genotypes number that fit to the schama *S* in iteration t
- For S_0= (*** 111 ***)

  $\xi(S_0,t)$ = 2 (v_1 and v_9 fit this schema)

***Pop*=10 *n*=9; 3*order-3 deceptive functions**

v_1= (101 **111** 000)
v_2= (000 100 011)
v_3= (111 101 000)
v_4= (010 000 101)
v_5= (001 000 111)
v_6= (000 001 000)
v_7= (000 100 000)
v_8= (011 000 001)
v_9= (000 **111** 000)
v_{10}= (011 100 011)

Schema theorem

- *eval(S,t)*
  - Fitness of schema *S* in iteration *t*
  - Average fitness of the genotypes that fit to schema *S*
  - Assume that *p* genotypes fits to schema *S*: $\{v_{S1}, v_{S2}, \ldots, v_{Sp}\}$
  - $eval(S,t) = \sum_{j=1}^{p} v_{Sj} \,/p$
- For S_0= (*** 111 ***)
  - *eval(S_0,t)* = (5+7)/2 = 6

***Pop*=10 *n*=9; 3*order-3 deceptive functions**

v_1= (101 **111** 000) 0+3+2=5
v_2= (000 100 011) 2+1+0=3
v_3= (111 101 000) 3+0+2=5
v_4= (010 000 101) 1+2+0=3
v_5= (001 000 111) 1+2+3=6
v_6= (000 001 000) 2+1+2=5
v_7= (000 100 000) 2+1+2=5
v_8= (011 000 001) 0+2+1=3
v_9= (000 **111** 000) 2+3+2=7
v_{10}= (011 100 011)0+1+0=1

Schema theorem

- Choose genotype that fits to schema *S* as a parent
  - **What is the probability?**
  - F(t) – summarize fitness
  - $F(t) = \sum_{j=1}^{pop} v_j = 43$
- For S_0= (*** 111 ***)

$\xi(S_0,t+1)$ =

$\xi(S_0,t) \cdot pop \cdot eval(S_0,t) / F(t)$ =

2·10·6/43 ≈ **2.79**

***Pop*=10 *n*=9; 3*order-3 deceptive functions**

v_1= (101 **111** 000) 0+3+2=5

v_2= (000 100 011) 2+1+0=3

v_3= (111 101 000) 3+0+2=5

v_4= (010 000 101) 1+2+0=3

v_5= (001 000 111) 1+2+3=6

v_6= (000 001 000) 2+1+2=5

v_7= (000 100 000) 2+1+2=5

v_8= (011 000 001) 0+2+1=3

v_9= (000 **111** 000) 2+3+2=7

v_{10}= (011 100 011)0+1+0=1

Schema theorem

- If we only do the selection, then:

$\xi(S_0,t+1) = \xi(S_0,t) \cdot pop \cdot eval(S_0,t) / F(t)$

- $\overline{F(t)} = F(t)/pop$ – average population fitness

$\xi(S_0,t+1) = \xi(S_0,t) \cdot eval(S_0,t) / \overline{F(t)}$

- For S_0= (*** 111 ***)
  - Fitness is above the average
  - Thus, if we only use selection - $\xi(S_0,t)$ → **will increase**

Schema theorem

- Crossover
  - We cross the **genotype that fits** to schema *S* with the **genotype that does not fit** to schema *S*
  - Will schema *S* survive this crossover?
  - **[simplification]** we assume that we have to pass all the genes from the genotype that fits schema *S*

Schema theorem

- For S_0= (*** 111 ***)
- 1-point crossover
  - 8 possible cross points
  - 2 point destroy the schema
  - The chance to **destroy** the schema:

$$P_d(S) = \frac{\delta(S)}{n-1} \qquad P_d(S_0) = \frac{6-4}{9-1} = \frac{2}{8} = 0.25$$

  - The chance to **preserve** the schema:

$$P_s(S) = 1 - \frac{\delta(S)}{n-1} \qquad P_s(S_0) = 1 - 0.25 = 0.75$$

Schema theorem

- For S_0= (*** 111 ***)
- *Uniform* crossover
  - Choose 3 genes from the parent that fits schema *S*
  - The chance to **preserve** the schema:

$$P_s(S) = 2^{-o(S)+1} \qquad P_s(S_0) = 2^{-3+1} = 0.25$$

Schema theorem

- Crossover executed with probability p_c
- The chance to **preserve** the schema after crossover:

$$P_s(S) = 1 - p_c \frac{\delta(S)}{n-1} \qquad \text{(1-point)}$$

$$P_s(S) = (1 - p_c) + p_c \cdot 2^{-o(S)+1} \qquad \text{(\textit{uniform})}$$

- If we cut the schema into 2 pieces – sometimes schema can survive it:
  - Cut on the 5th position.: (000 111 000) i (101 101 111)
  - We get: (000 11+1 111) i (101 10+1 000)
- Thus: $\boldsymbol{P_s(S) \geq 1 - p_c \frac{\delta(S)}{n-1}}$

Schema theorem

- For S_0= (*** 111 ***); $p_c = 0.3$

$\xi(S_0,t+1) = \xi(S_0,t)\cdot eval(S_0,t) \,/\, \overline{F(t)}\cdot P_{c,S}$

For 1-point:

$\xi(S_0,t+1) \geq \xi(S_0,t)\cdot eval(S_0,t) \,/\, \overline{F(t)}\cdot[1 - p_c \frac{\delta(S)}{n-1}]=$

$2\cdot 6/4{,}3 \cdot [1 - 0.3 \cdot \frac{6-4}{9-1}] \approx 2.79 \cdot [1 - 0.075] \approx$ **2.58**

Schema theorem

- For S_0= (*** 111 ***); $p_c = 0.3$

$\xi(S_0,t+1) = \xi(S_0,t)\cdot eval(S_0,t) \,/\, \overline{F(t)}\cdot P_{c,S}$

For *uniform*:

$\xi(S_0,t+1) \geq \xi(S_0,t)\cdot eval(S_0,t) \,/\, \overline{F(t)}\cdot[(1 - p_c) + p_c \cdot 2^{-o(S)+1}]=$

$2\cdot 6/4{,}3 \cdot[\,(1 - 0.3) + 0.3 \cdot 2^{-o(S)+1}] \approx$

$2.79 \cdot [0.7 + 0.3 \cdot 0.25] = 2.79 \cdot 0.775 \approx$ **2.16**

Schema theorem

- The chance to **survive** mutation:
- p_m- mutation probability
- The chance to **survive** mutation:

$$P_{s,m}(S) = \left(1 - p_m\right)^{o(S)}$$

- Because p_m << 1

$$P_{s,m}(S) \approx 1 - o(S) \cdot p_m$$

- For S_0= (*** 111 ***); $p_c = 0.01$

$$P_{s,m}(S) \approx 1 - 3 \cdot 0.01 \quad = 0.97$$

Schema theorem

- For 1-point crossover:

$\xi(S_0,t+1) \geq$

$\xi(S_0,t) \cdot eval(S_0,t) / \overline{F(t)} \cdot [1 - p_c \frac{\delta(S)}{n-1} - o(S) \cdot p_m]$

(Dla S_0 to: **2.50**)

- For *uniform* crossover:

$\xi(S_0,t+1) \geq$

$\xi(S_0,t) \cdot eval(S_0,t) / \overline{F(t)} \cdot [(1 - p_c) + p_c \cdot 2^{-o(S)+1} - o(S) \cdot p_m]$

(Dla S_0 to: **2.08**)

Schema theorem

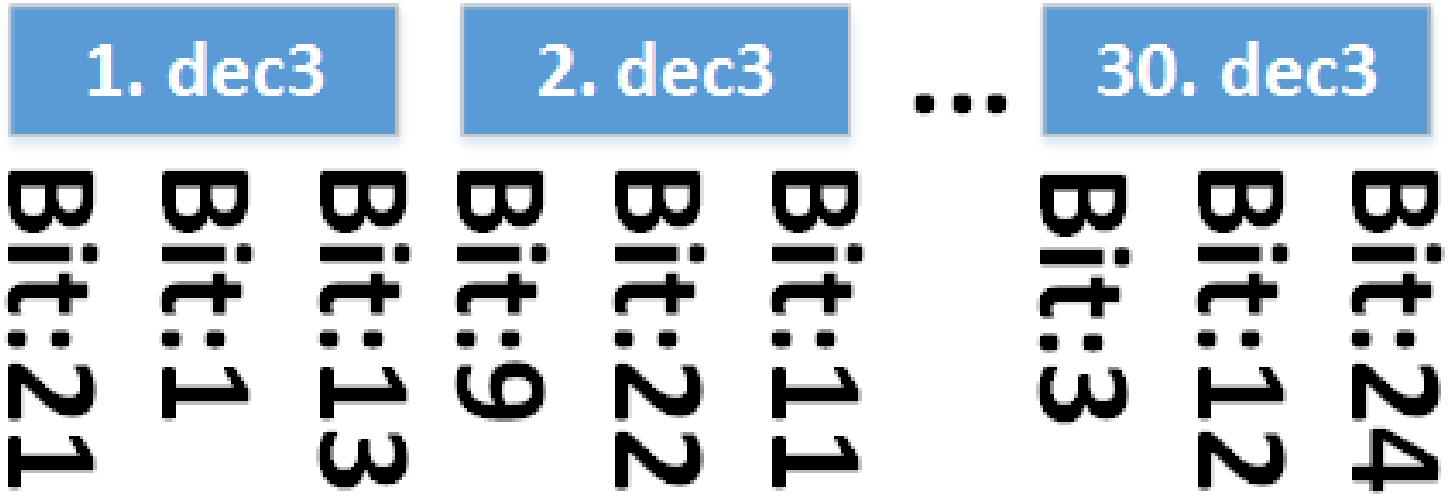
- Assumes the population is huge
- Only simple crossover types
- Yet it supports the main intuition:
  - **The number of short schema with the fitness above the average will increase**

Schema theorem

- Schema theorem – can we **REALLY** compute something?
- **No, we can not**…
- **…then why it is useful?**
  - You can make a lecture about it and earn money…
  - It supports **the right** intuitions
  - It helps to raise the right question:
    - How to exchange the schema that have a high defining length but low order?
  - Conclusions – problem decomposition (next lecture)

Concatenation with shuffled genes

- **Let's get back to the main question:**



Why 1-point crossover have lost the effectiveness?
Some blocks become long…
1-point crossover can not proces them

Do you understand it? That's what the lecture was for!!!

Genetic Algorithms

- Why *uniform* crossover worked better than the 1-point?

  It is not sensitive to the gene order

- Why *uniform* crossover worked poorly?

Because it can only process schemata with a very low number of genes

- **What kind of crossover do we need?**

– **Gene position - irrelevant**

– **Can proces long schemata…**