Intelligens Számítási Módszerek
Genetikus algoritmusok, gradiens mentes optimálási módszerek

2005/2006. tanév, II. félév

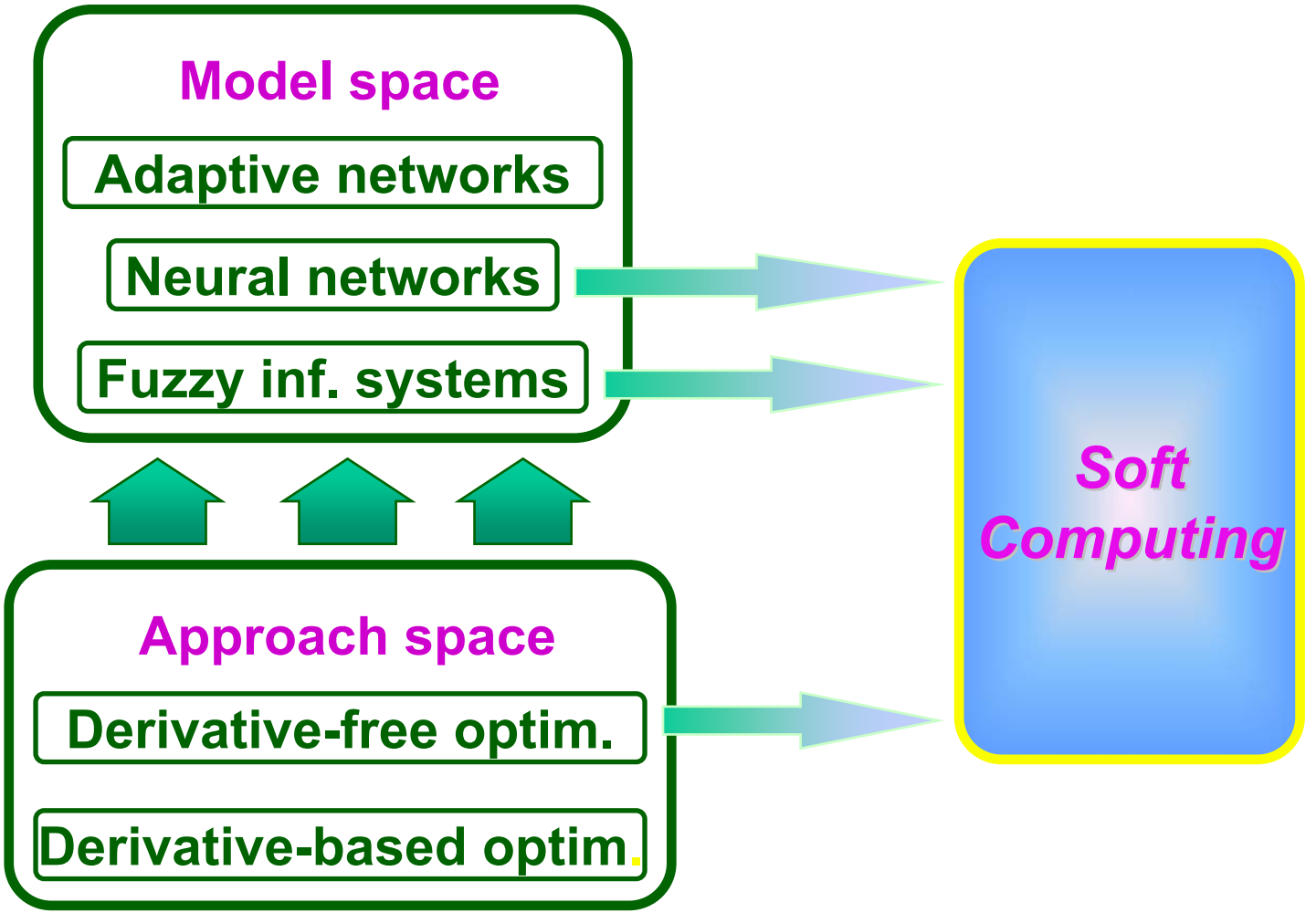
Dr. Kovács Szilveszter

E-mail: szkovacs@iit.uni-miskolc.hu

Informatikai Intézet 106. sz. szoba

Tel: (46) 565-111 / 21-06

Soft Computing - The Big Picture



Can evolution be intelligent?

- Intelligence can be defined as the capability of a system to adapt its behaviour to ever-changing environment.
- According to Alan Turing, the form or appearance of a system is irrelevant to its intelligence.
- Evolutionary computation simulates evolution on a computer. The result of such a simulation is a series of optimisation algorithms, usually based on a simple set of rules.
- Optimisation iteratively improves the quality of solutions until an optimal, or at least feasible, solution is found.

Can evolution be intelligent?

- The behaviour of an individual organism is an inductive inference about some yet unknown aspects of its environment. If, over successive generations, the organism survives, we can say that this organism is capable of learning to predict changes in its environment.
- The evolutionary approach is based on computational models of natural selection and genetics.
- We call them **evolutionary computation**, an umbrella term that combines **genetic algorithms**, **evolution strategies** and **genetic programming**.

Simulation of natural evolution

- On 1 July 1858, **Charles Darwin** presented his theory of evolution before the Linnean Society of London. This day marks the beginning of a revolution in biology.
- Darwin's classical **theory of evolution**, together with Weismann's **theory of natural selection** and Mendel's concept of **genetics**, now represent the neo-Darwinian paradigm.

Simulation of natural evolution

- **Neo-Darwinism** is based on processes of reproduction, mutation, competition and selection.
- The power to reproduce appears to be an essential property of life.
- The power to mutate is also guaranteed in any living organism that reproduces itself in a continuously changing environment.
- Processes of competition and selection normally take place in the natural world, where expanding populations of different species are limited by a finite space.

Simulation of natural evolution

- Evolution can be seen as a process leading to the maintenance of a population's ability to survive and reproduce in a specific environment. This ability is called **evolutionary fitness**.
- Evolutionary fitness can also be viewed as a measure of the organism's ability to anticipate changes in its environment.
- The **fitness**, or the quantitative measure of the ability to predict environmental changes and respond adequately, can be considered as the **quality** that is optimised in natural life.

How is a population with increasing fitness generated?

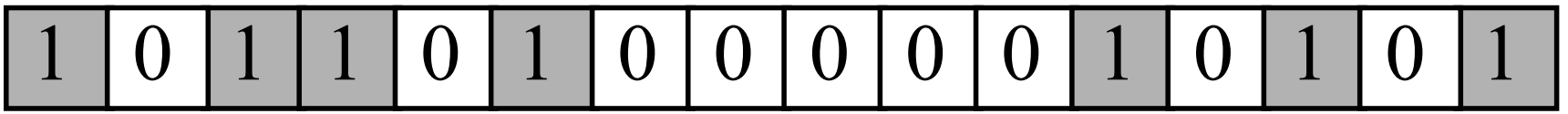
- Let us consider a population of rabbits. Some rabbits are faster than others, and we may say that these rabbits possess superior fitness, because they have a greater chance of avoiding foxes, surviving and then breeding.
- If two parents have superior fitness, there is a good chance that a combination of their genes will produce an offspring with even higher fitness. Over time the entire population of rabbits becomes faster to meet their environmental challenges in the face of foxes.

Simulation of natural evolution

- All methods of evolutionary computation simulate natural evolution by creating a population of individuals, evaluating their fitness, generating a new population through genetic operations, and repeating this process a number of times.
- We will start with **Genetic Algorithms** (GAs) as most of the other evolutionary algorithms can be viewed as variations of genetic algorithms.

Genetic Algorithms

- In the early 1970s, John Holland introduced the concept of genetic algorithms.
- His aim was to make computers do what nature does. Holland was concerned with algorithms that manipulate strings of binary digits.
- Each artificial "chromosomes" consists of a number of "genes", and each gene is represented by 0 or 1:



Genetic Algorithms

- Nature has an ability to adapt and learn without being told what to do. In other words, nature finds good chromosomes blindly. GAs do the same.
- Two mechanisms link a GA to the problem it is solving: **encoding** and **evaluation**.
- The GA uses a measure of fitness of individual chromosomes to carry out reproduction. As reproduction takes place, the crossover operator exchanges parts of two single chromosomes, and the mutation operator changes the gene value in some randomly chosen location of the chromosome.

Basic genetic algorithms

Step 1: Represent the problem variable domain as a chromosome of a fixed length, choose the size of a chromosome population N, the crossover probability p_c and the mutation probability p_m.

Step 2: Define a fitness function to measure the performance, or fitness, of an individual chromosome in the problem domain. The fitness function establishes the basis for selecting chromosomes that will be mated during reproduction.

Basic genetic algorithms

Step 3: Randomly generate an initial population of chromosomes of size N:

$x_1, x_2, \ldots, x_N$

Step 4: Calculate the fitness of each individual chromosome:

$f(x_1), f(x_2), \ldots, f(x_N)$

Step 5: Select a pair of chromosomes for mating from the current population. Parent chromosomes are selected with a probability related to their fitness.

Step 6: Create a pair of offspring chromosomes by applying the genetic operators – **crossover** and **mutation**.

Step 7: Place the created offspring chromosomes in the new population.

Step 8: Repeat *Step 5* until the size of the new chromosome population becomes equal to the size of the initial population, *N*.

Step 9: Replace the initial (parent) chromosome population with the new (offspring) population.

Step 10: Go to *Step 4*, and repeat the process until the termination criterion is satisfied.

Genetic algorithms

- GA represents an iterative process. Each iteration is called a **generation**. A typical number of generations for a simple GA can range from 50 to over 500. The entire set of generations is called a **run**.
- Because GAs use a stochastic search method, the fitness of a population may remain stable for a number of generations before a superior chromosome appears.
- A common practice is to terminate a GA after a specified number of generations and then examine the best chromosomes in the population. If no satisfactory solution is found, the GA is restarted.

Genetic algorithms: case study

A simple example will help us to understand how a GA works. Let us find the maximum value of the function ($15x - x^2$) where parameter x varies between 0 and 15. For simplicity, we may assume that x takes only integer values. Thus, chromosomes can be built with only four genes:

Integer	*Binary code*	*Integer*	*Binary code*	*Integer*	*Binary code*
1	0 0 0 1	6	0 1 1 0	11	1 0 1 1
2	0 0 1 0	7	0 1 1 1	12	1 1 0 0
3	0 0 1 1	8	1 0 0 0	13	1 1 0 1
4	0 1 0 0	9	1 0 0 1	14	1 1 1 0
5	0 1 0 1	10	1 0 1 0	15	1 1 1 1

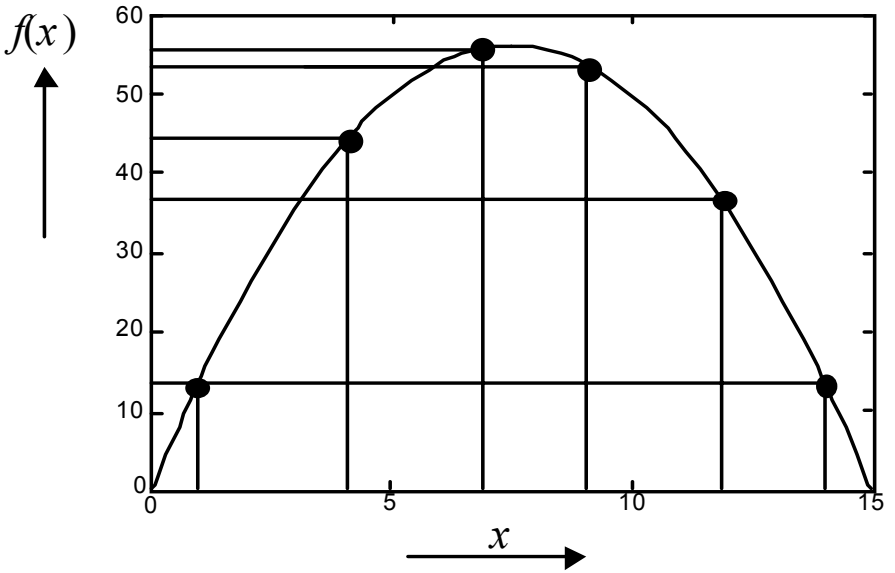
Genetic algorithms: case study

Suppose that the size of the chromosome population N is 6, the crossover probability p_c equals 0.7, and the mutation probability p_m equals 0.001. The fitness function in our example is defined by

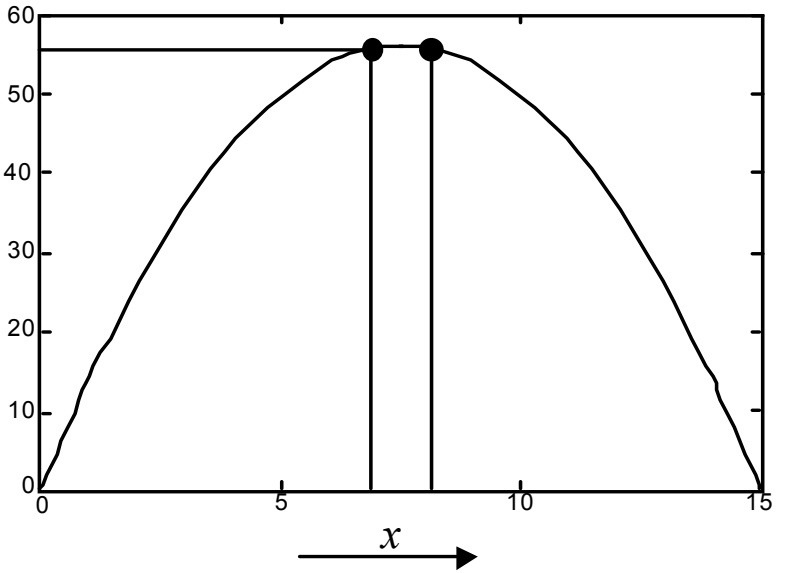
$$f(x) = 15\,x - x^2$$

The fitness function and chromosome locations

Chromosome label	*Chromosome string*	*Decoded integer*	*Chromosome fitness*	*Fitness ratio, %*
X1	1 1 0 0	12	36	16.5
X2	0 1 0 0	4	44	20.2
X3	0 0 0 1	1	14	6.4
X4	1 1 1 0	14	14	6.4
X5	0 1 1 1	7	56	25.7
X6	1 0 0 1	9	54	24.8



(*a*) Chromosome initial locations.



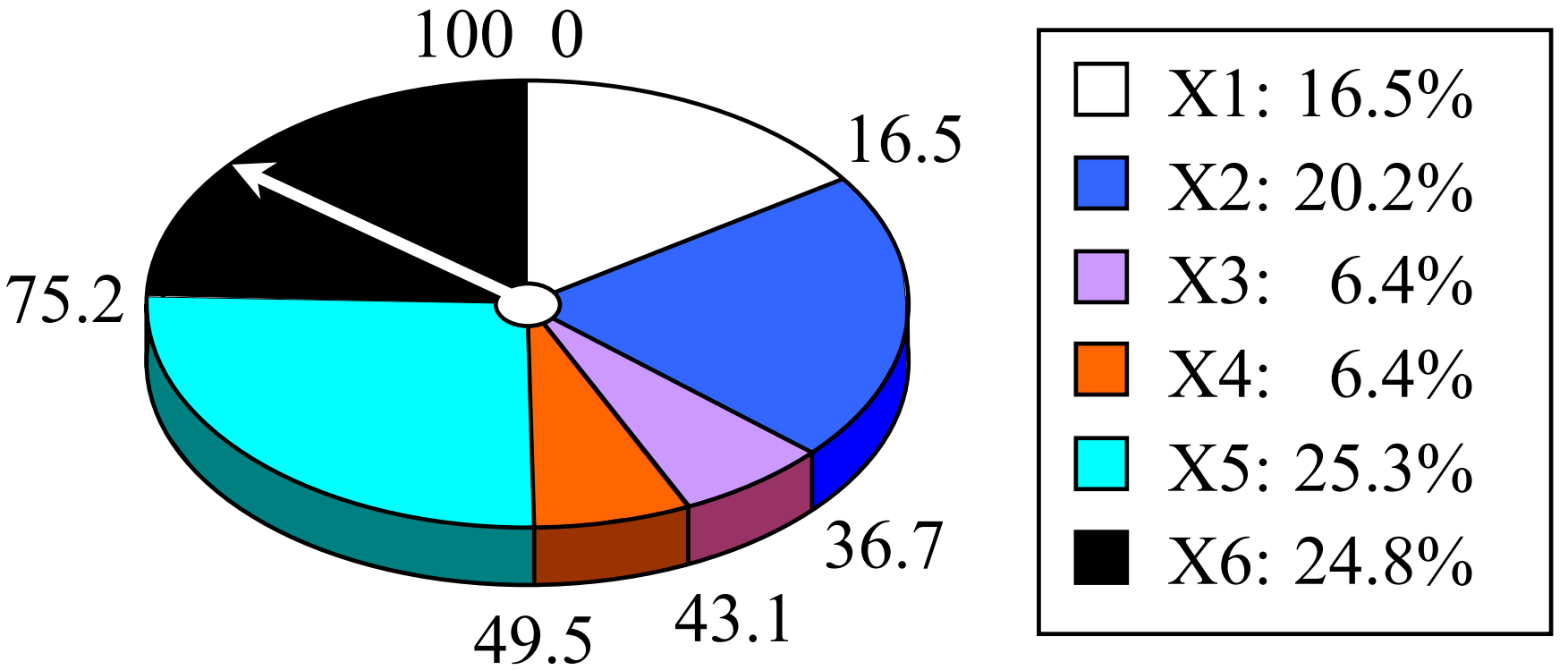
(*b*) Chromosome final locations.

Genetic algorithms

- In natural selection, only the fittest species can survive, breed, and thereby pass their genes on to the next generation. GAs use a similar approach, but unlike nature, the size of the chromosome population remains unchanged from one generation to the next.
- The last column in Table shows the ratio of the individual chromosome's fitness to the population's total fitness. This ratio determines the chromosome's chance of being selected for mating. The chromosome's average fitness improves from one generation to the next.

Roulette wheel selection

The most commonly used chromosome selection techniques is the **roulette wheel selection**.



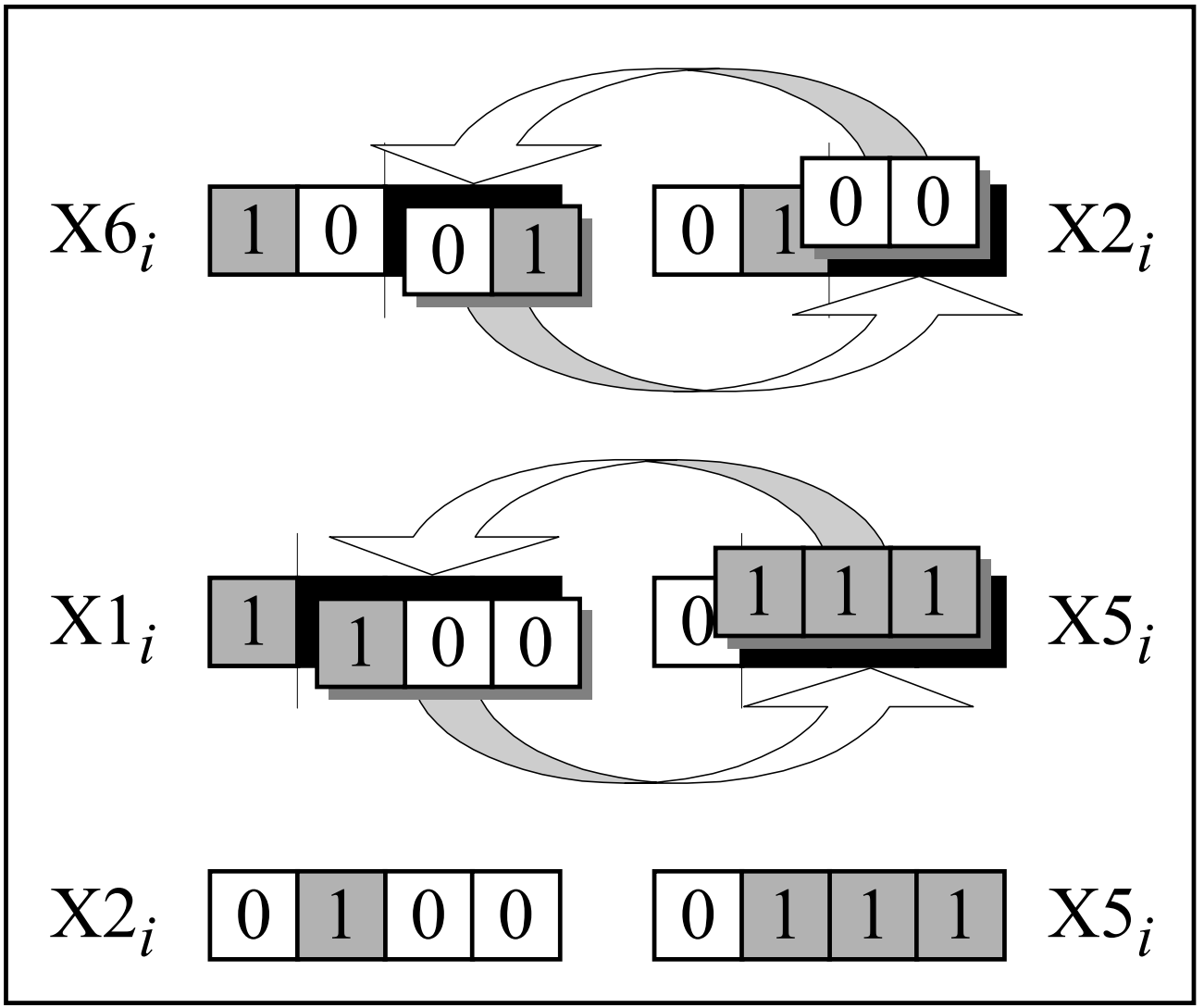
Crossover operator

- In our example, we have an initial population of 6 chromosomes. Thus, to establish the same population in the next generation, the roulette wheel would be spun six times.
- Once a pair of parent chromosomes is selected, the **crossover** operator is applied.

Crossover operator

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

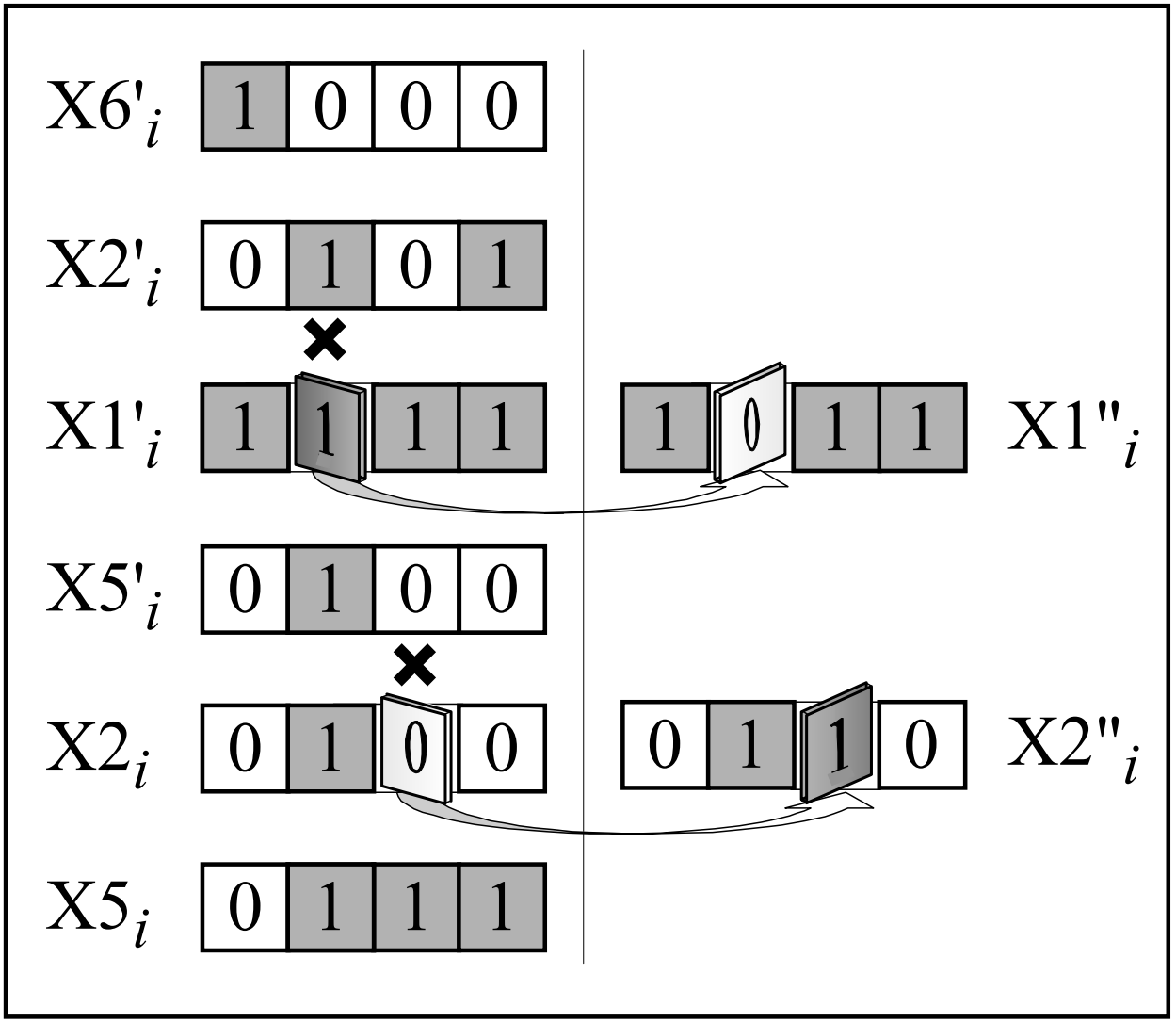
Crossover



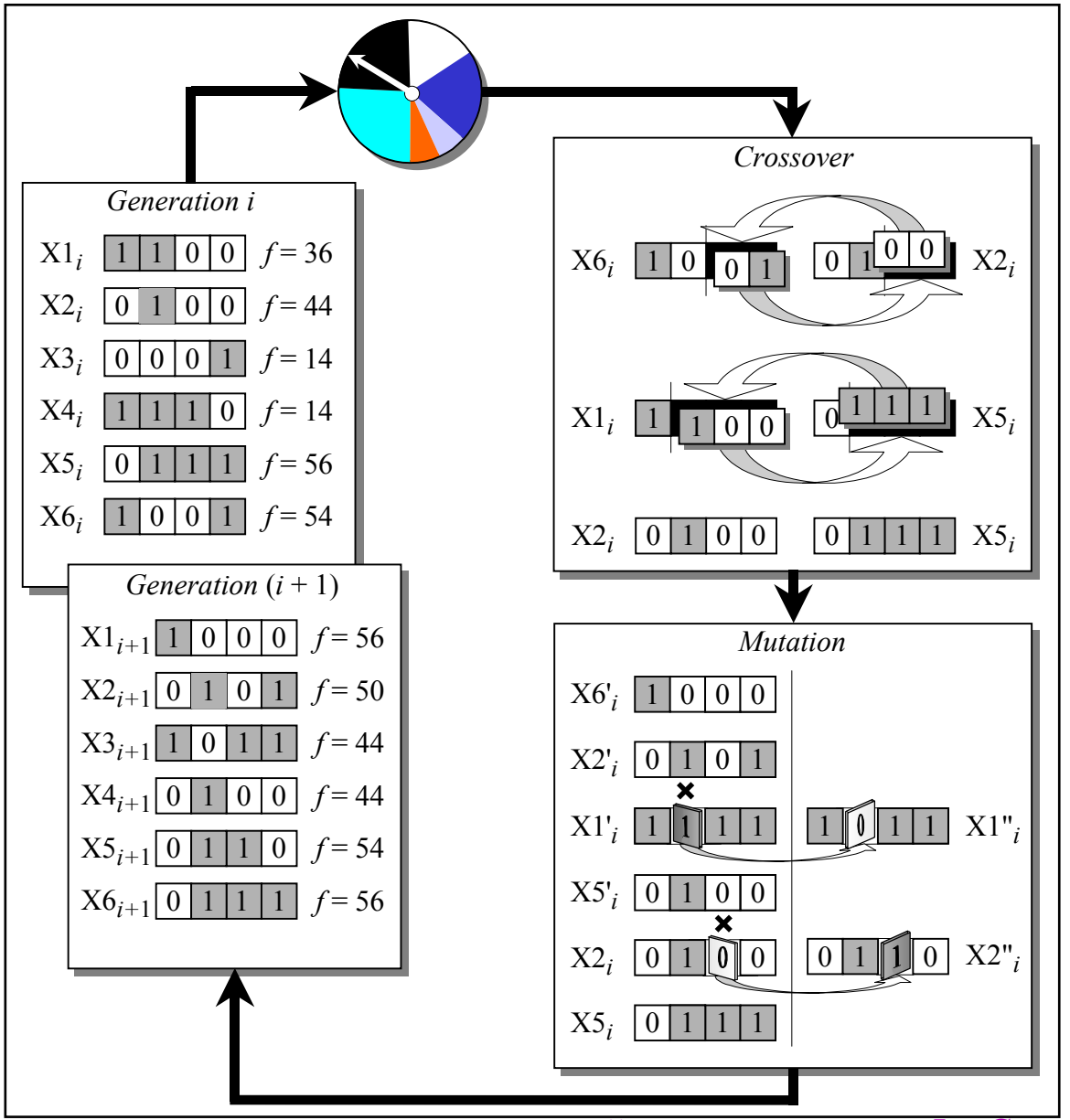
Mutation operator

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

Mutation


X6'i
1 0 0 0
X2'i
0 1 0 1
X1'i
1 1 1 1
1 0 1 1
X1"i
X5'i
0 1 0 0
X2i
0 1 0 0
0 1 1 0
X2"i
X5i
0 1 1 1

The genetic algorithm cycle

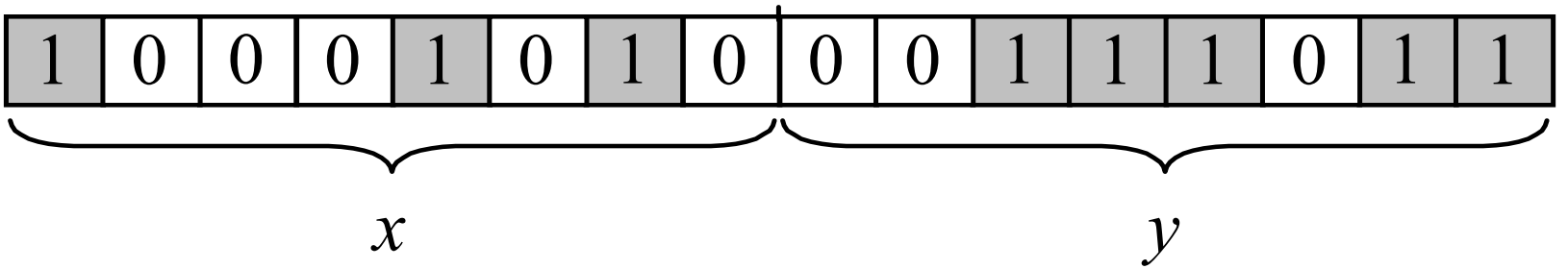


Genetic algorithms: case study

- Suppose it is desired to find the maximum of the "peak" function of two variables:

$$f(x,y) = (1-x)^2 e^{-x^2-(y+1)^2} - (x - x^3 - y^3)\, e^{-x^2-y^2}$$

- where parameters x and y vary between -3 and 3.
- The first step is to represent the problem variables as a chromosome – parameters x and y as a concatenated binary string:



Genetic algorithms: case study

- We also choose the size of the chromosome population, for instance 6, and randomly generate an initial population.
- The next step is to calculate the fitness of each chromosome. This is done in two stages.
- First, a chromosome, that is a string of 16 bits, is partitioned into two 8-bit strings:

1	0	0	0	1	0	1	0

and

0	0	1	1	1	0	1	1

- Then these strings are converted from binary (base 2) to decimal (base 10):

$$(10001010)_2 = 1\times 2^7 + 0\times 2^6 + 0\times 2^5 + 0\times 2^4 + 1\times 2^3 + 0\times 2^2 + 1\times 2^1 + 0\times 2^0 = (138)_{10}$$

and

$$(00111011)_2 = 0\times 2^7 + 0\times 2^6 + 1\times 2^5 + 1\times 2^4 + 1\times 2^3 + 0\times 2^2 + 1\times 2^1 + 1\times 2^0 = (59)_{10}$$

Genetic algorithms: case study

- Now the range of integers that can be handled by 8-bits, that is the range from 0 to $(2^8 - 1)$, is mapped to the actual range of parameters x and y, that is the range from –3 to 3:

$$\frac{6}{256-1} = 0.0235294$$

- To obtain the actual values of x and y, we multiply their decimal values by 0.0235294 and subtract 3 from the results:

$$x = (138)_{10} \times 0.0235294 - 3 = 0.2470588$$

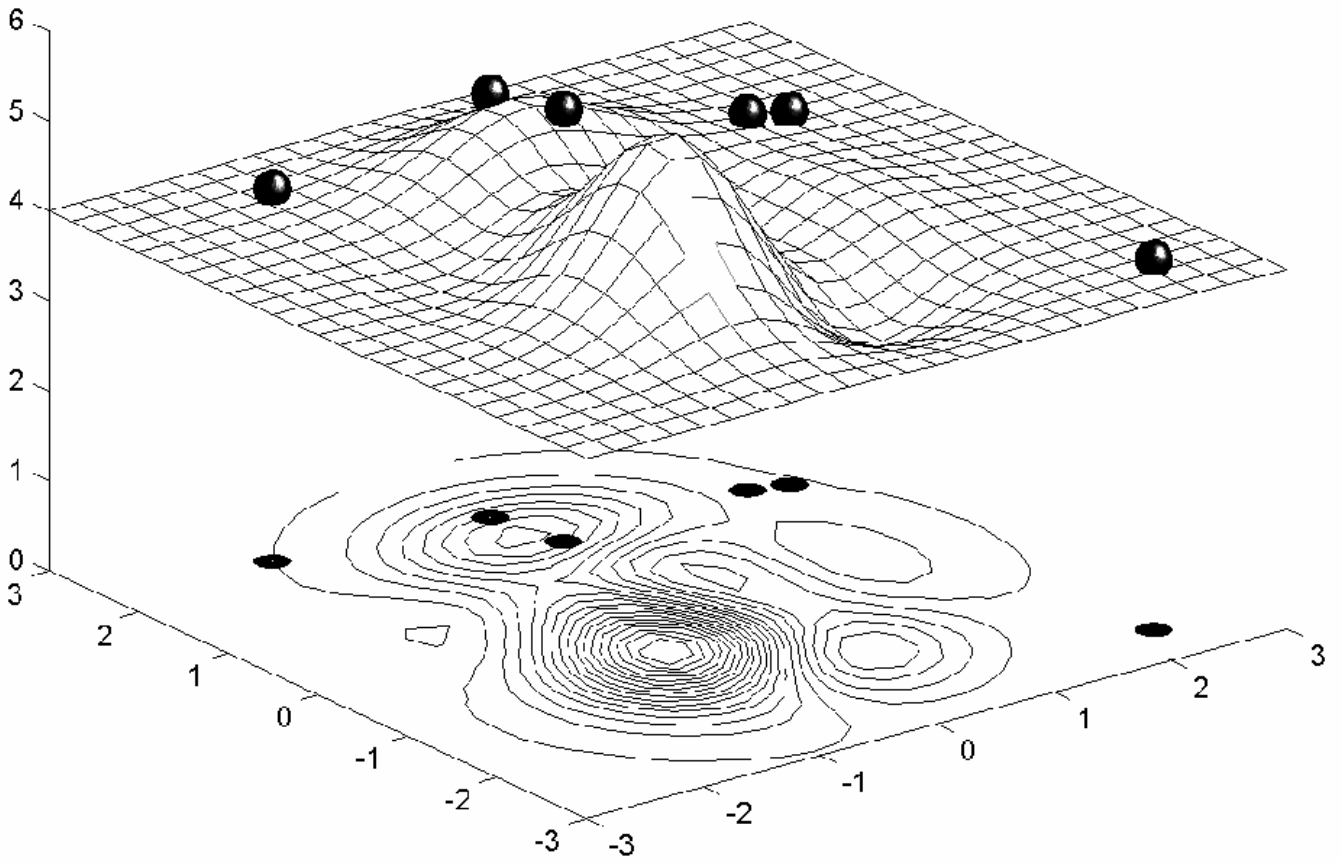
and

$$y = (59)_{10} \times 0.0235294 - 3 = -1.6117647$$

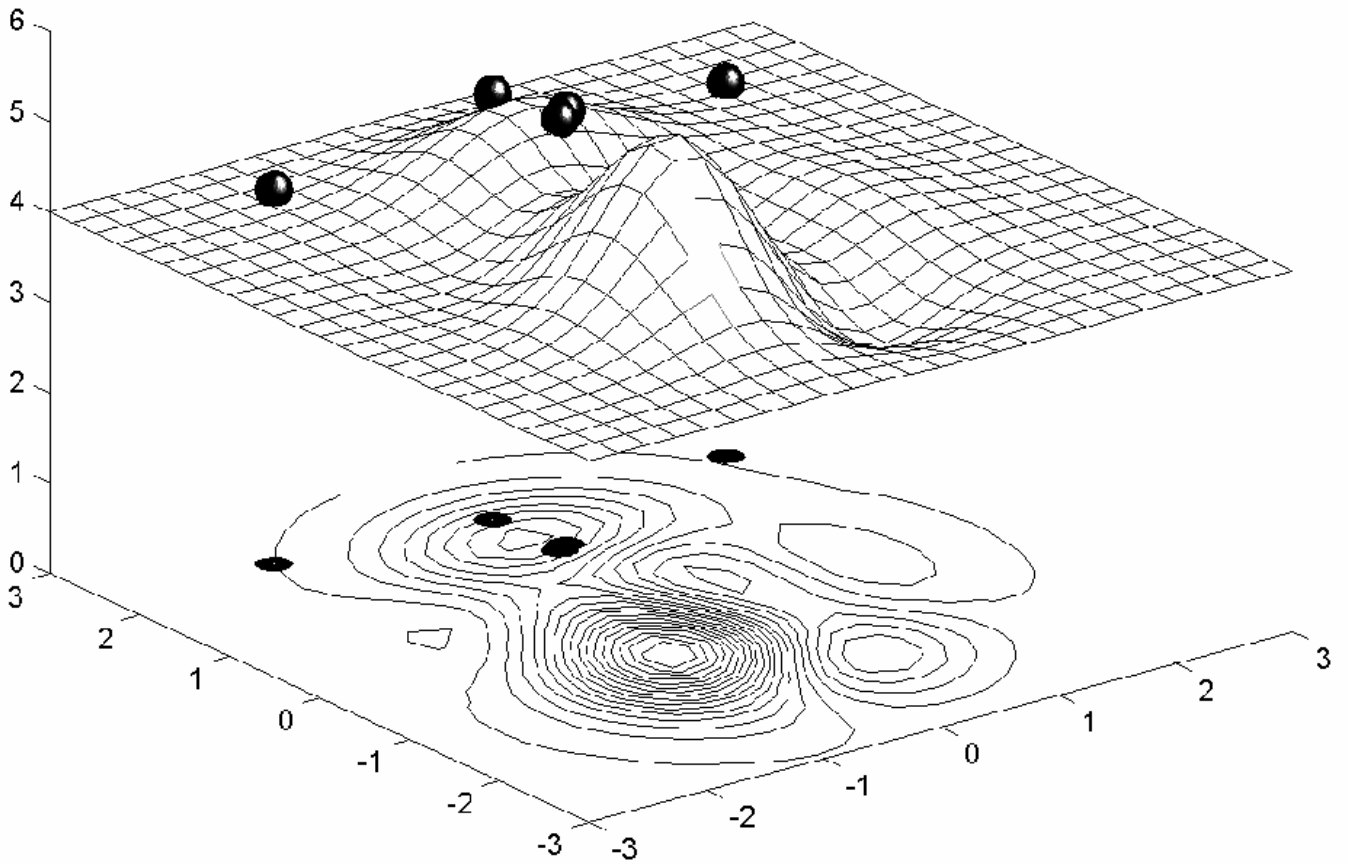
Genetic algorithms: case study

- Using decoded values of x and y as inputs in the mathematical function, the GA calculates the fitness of each chromosome.
- To find the maximum of the "peak" function, we will use crossover with the probability equal to 0.7 and mutation with the probability equal to 0.001. As we mentioned earlier, a common practice in GAs is to specify the number of generations. Suppose the desired number of generations is 100. That is, the GA will create 100 generations of 6 chromosomes before stopping.

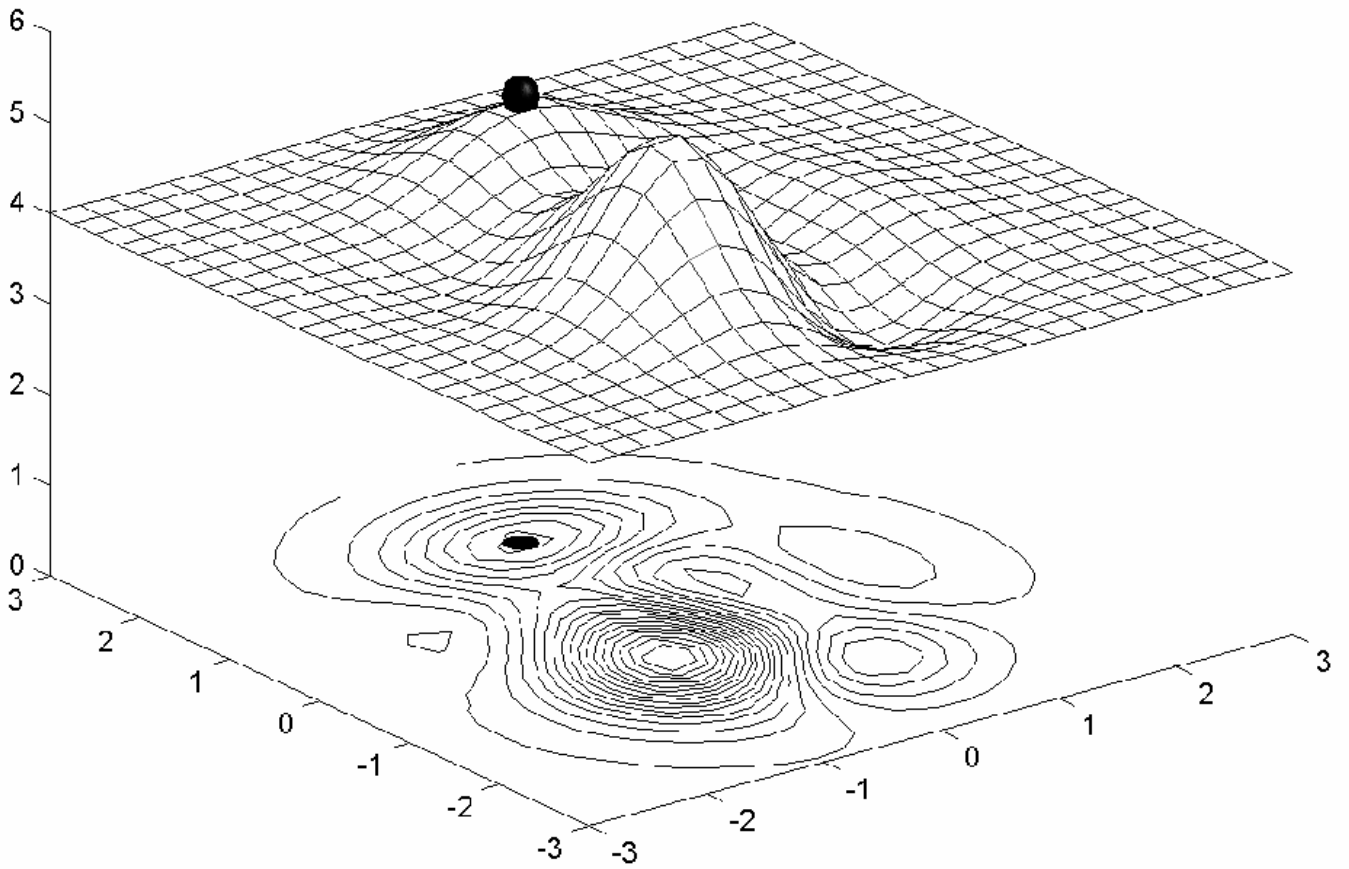
Chromosome locations on the surface of the "peak" function: initial population



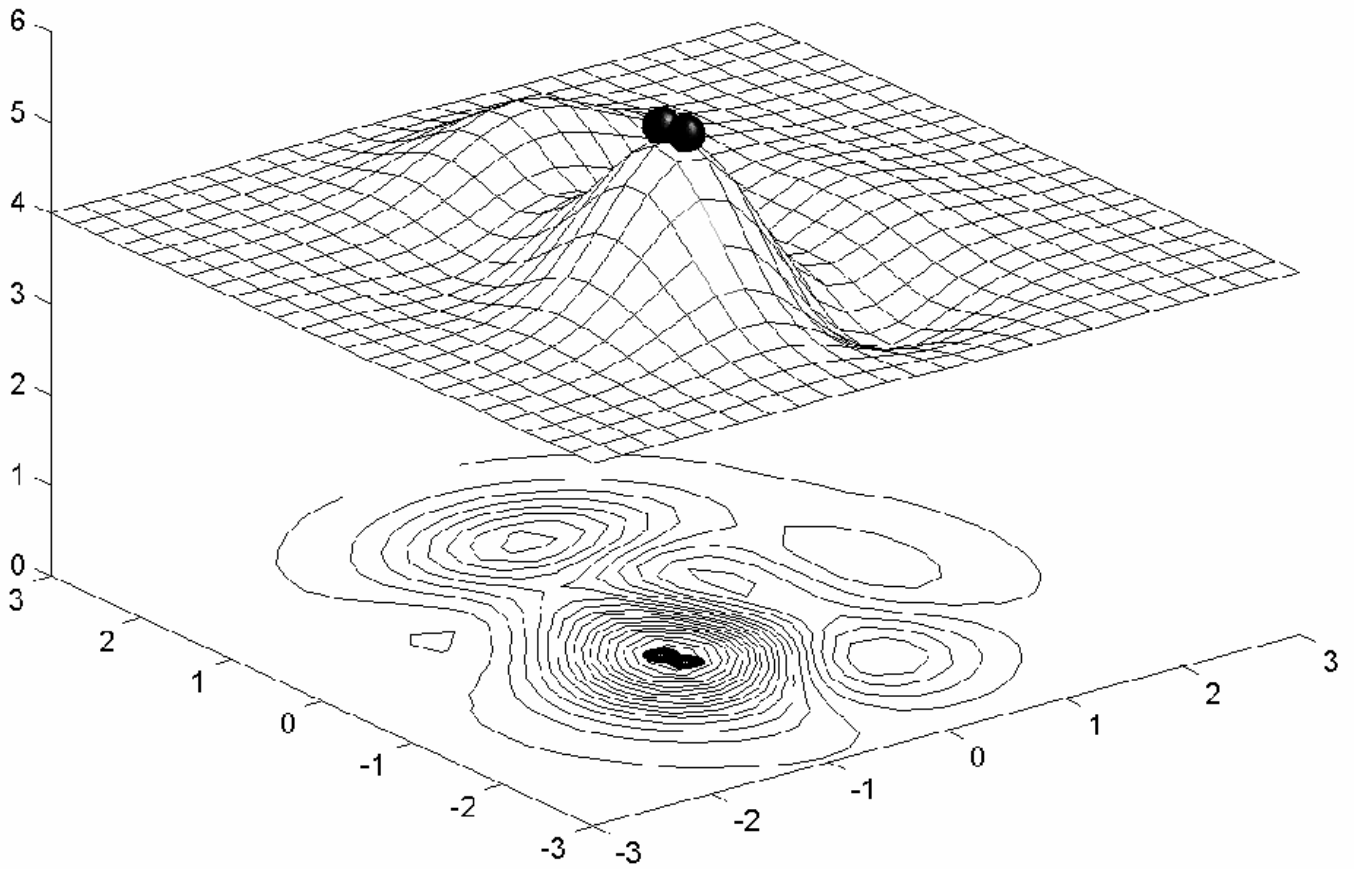
Chromosome locations on the surface of the "peak" function: first generation



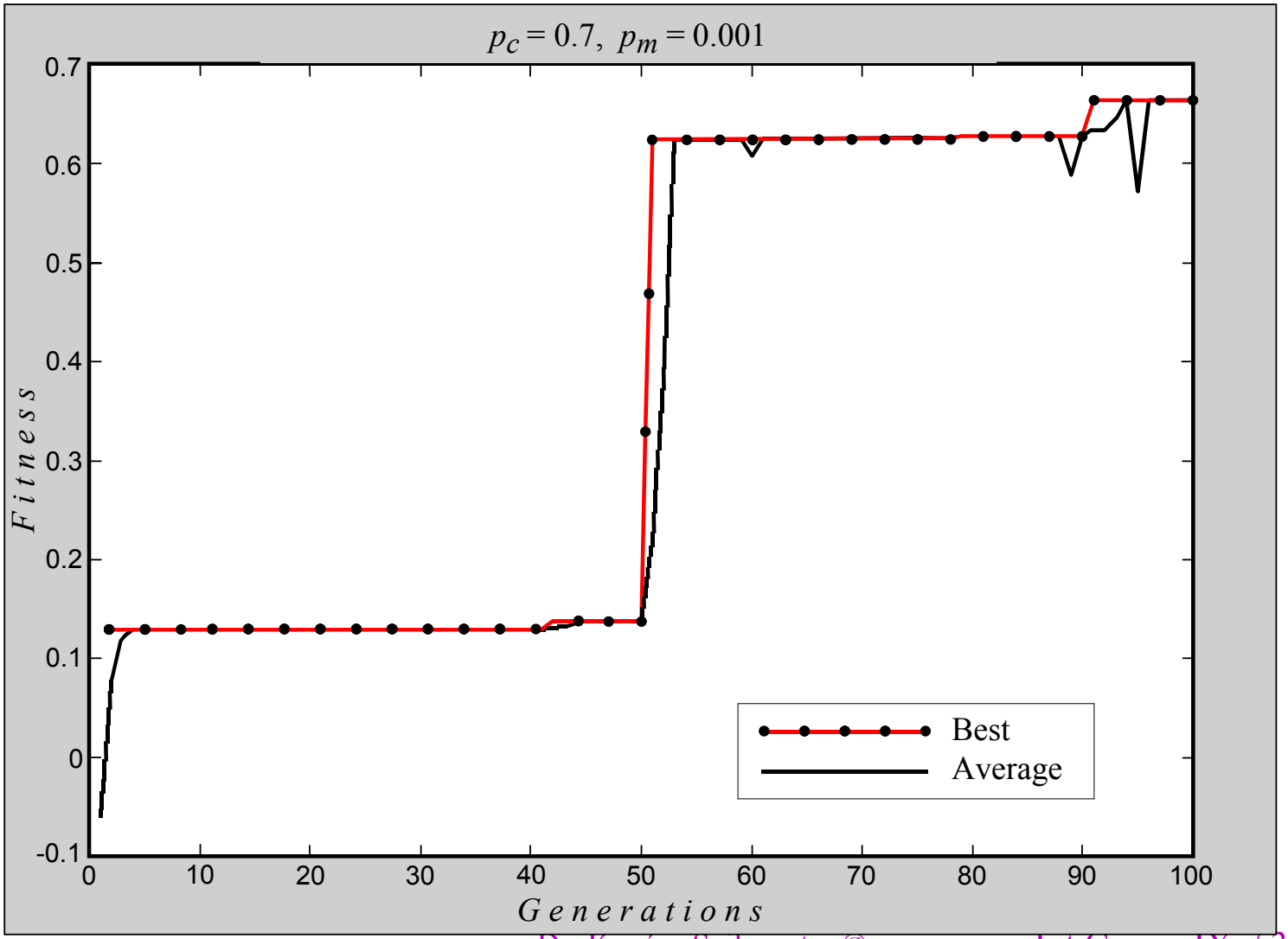
Chromosome locations on the surface of the "peak" function: local maximum



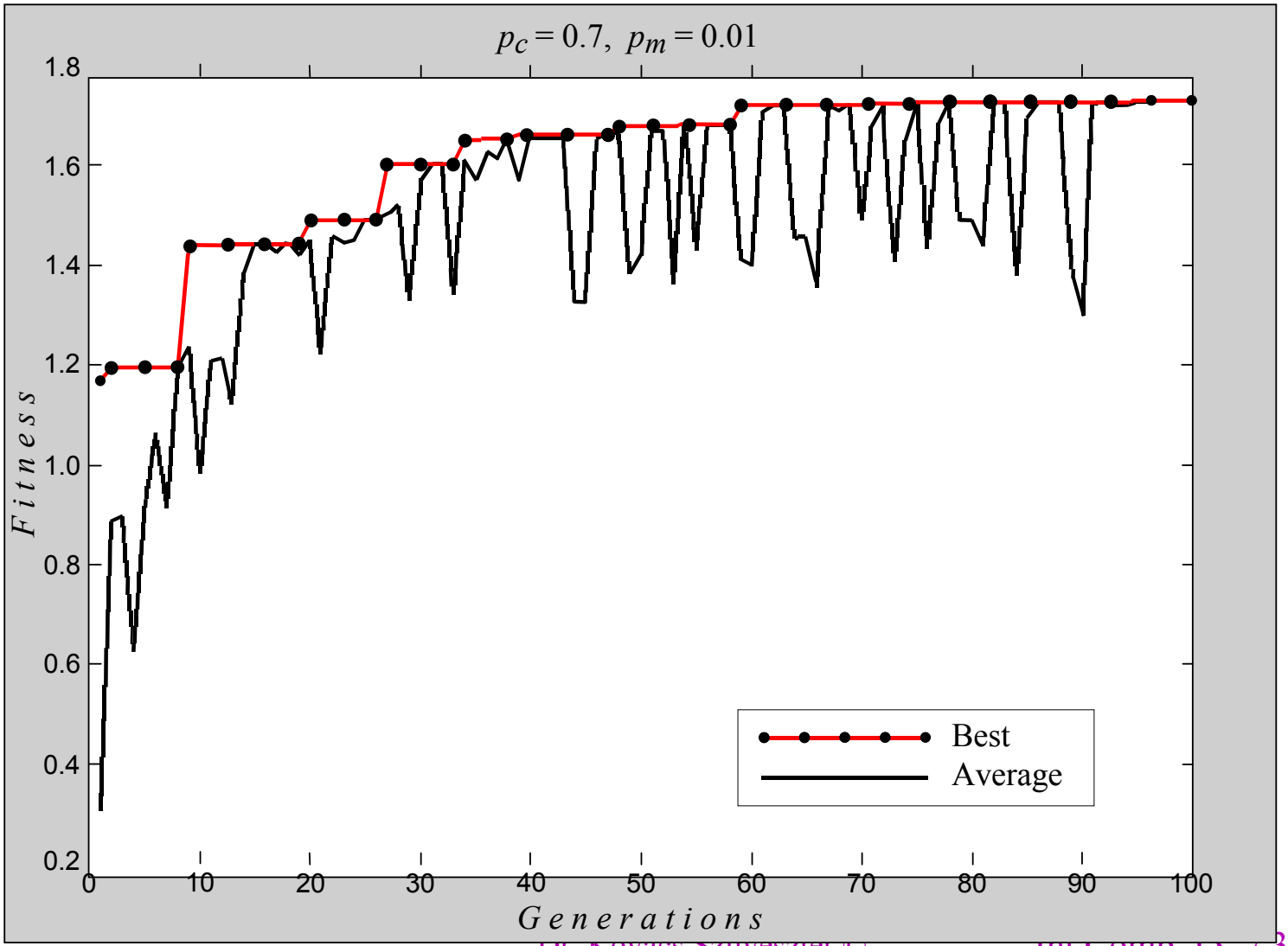
Chromosome locations on the surface of the "peak" function: global maximum



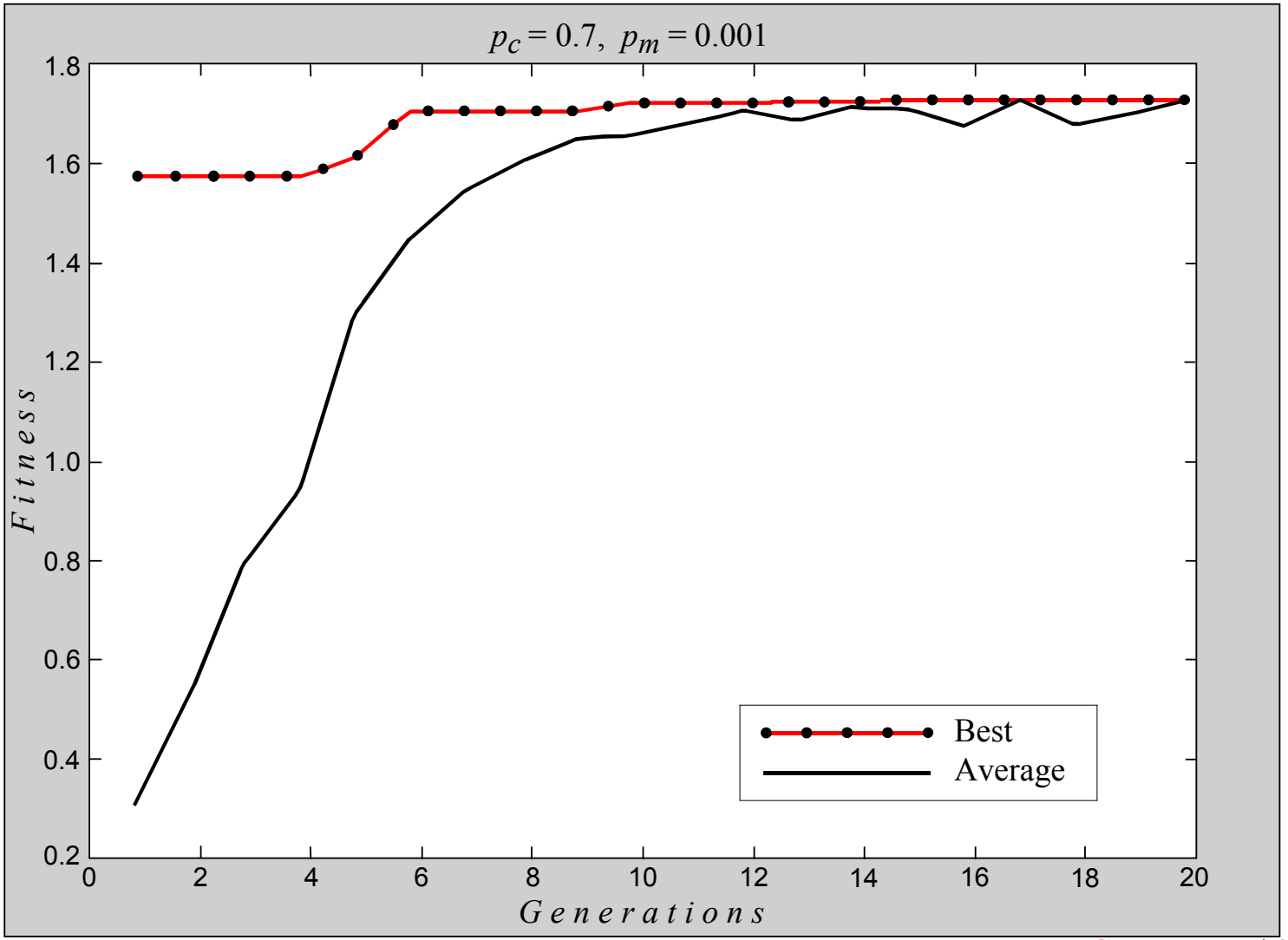
Performance graphs for 100 generations of 6 chromosomes: local maximum



Performance graphs for 100 generations of 6 chromosomes: global maximum



Performance graphs for 20 generations of 60 chromosomes

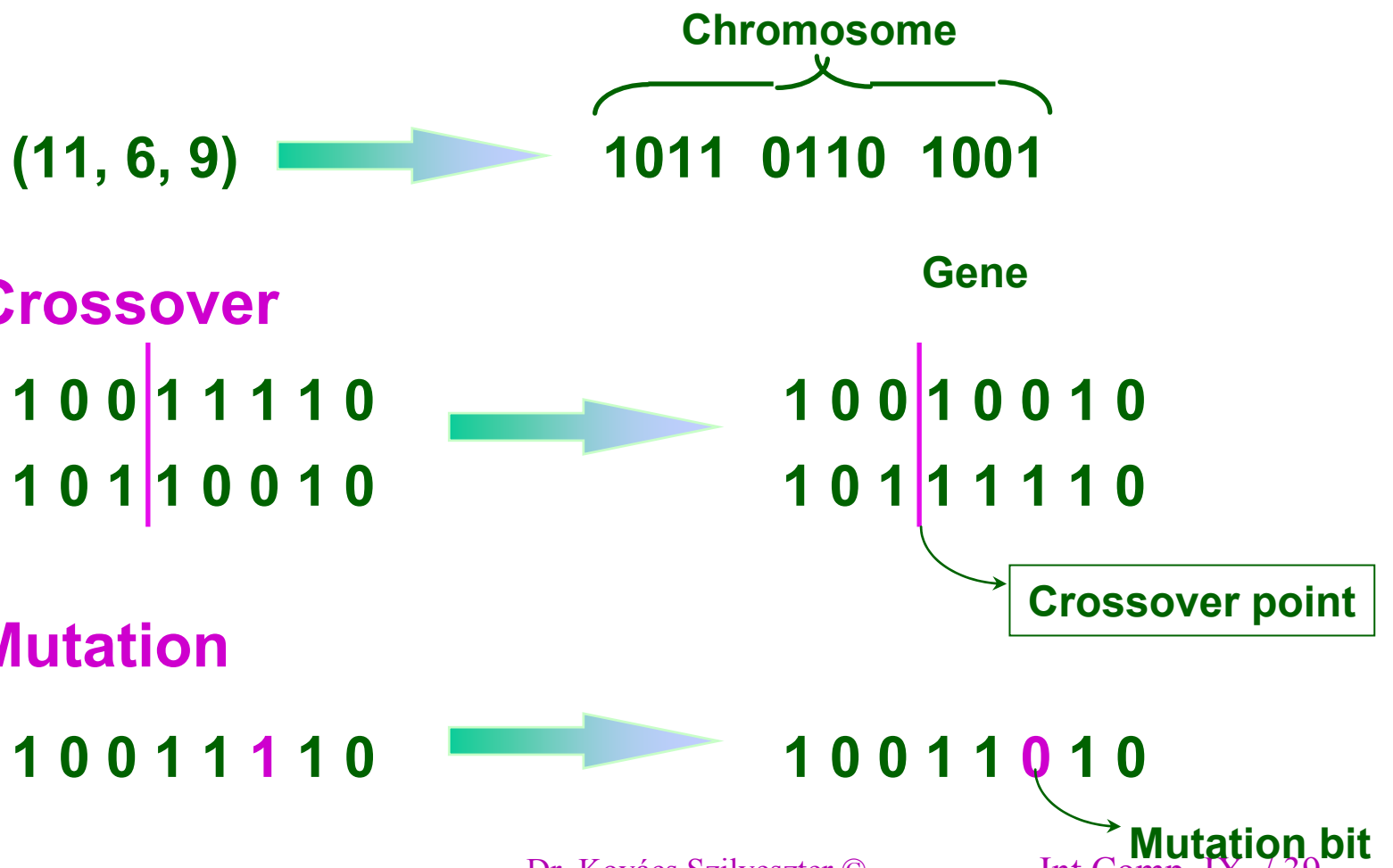


Genetic Algorithms - Summary

- **Terminology:**
  - **Fitness function**
  - **Population**
  - **Encoding schemes**
  - **Selection**
  - **Crossover**
  - **Mutation**
  - **Elitism**

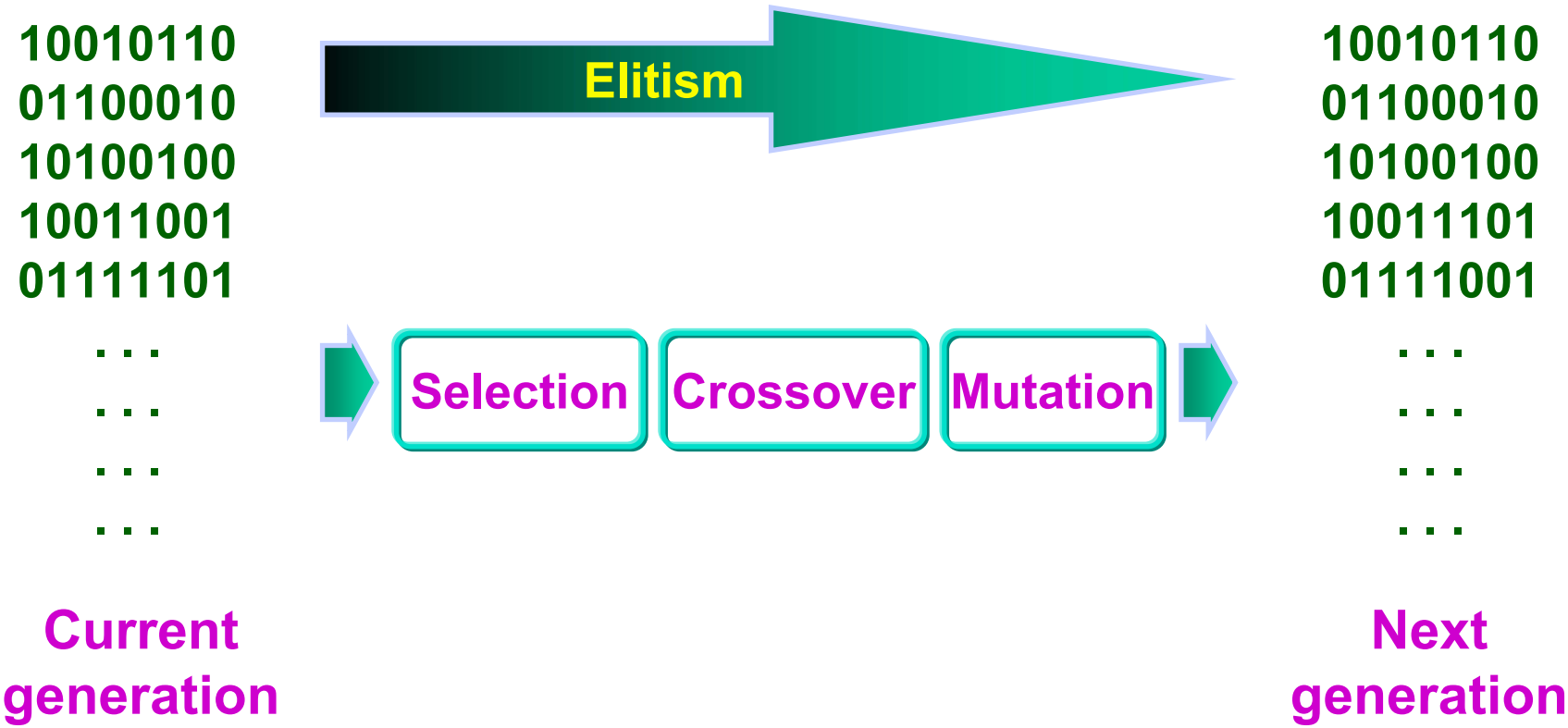
Genetic Algorithms - Summary

- **Binary encoding**



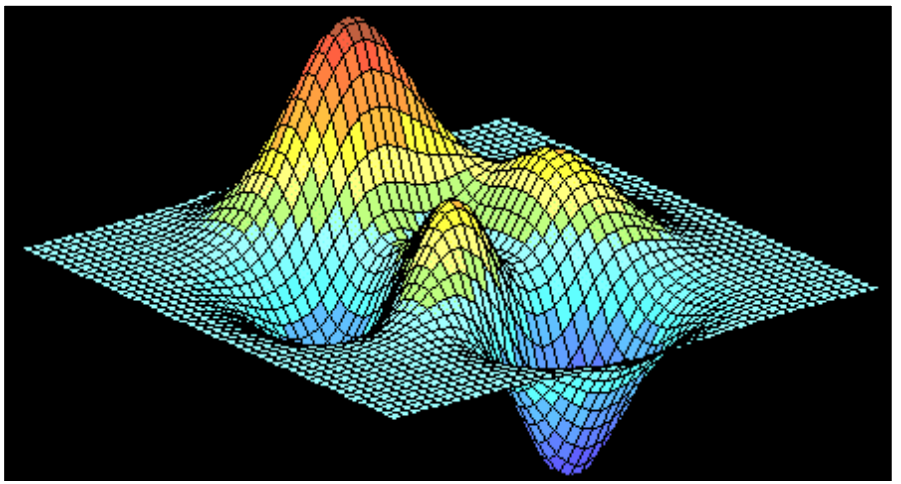
Genetic Algorithms - Summary

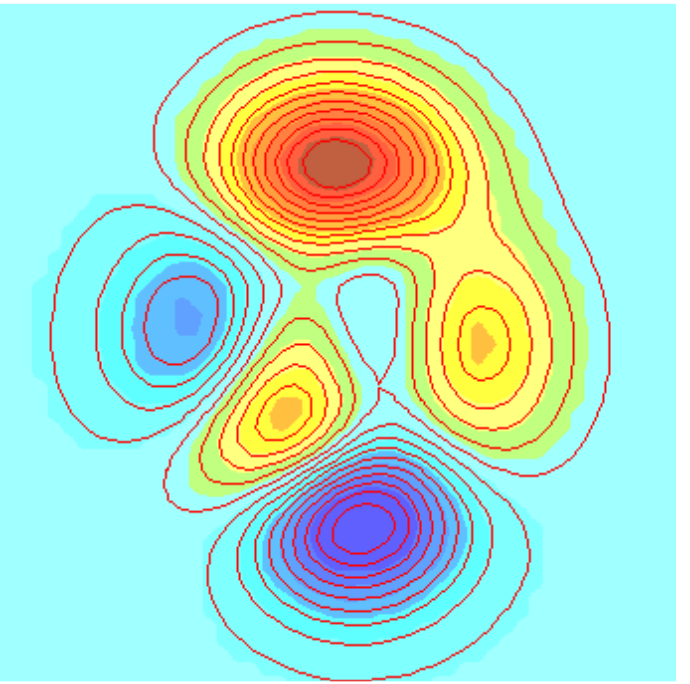
- **Flowchart**



Genetic Algorithms - Summary

- **Example: Find the max. of the "peaks" function**
- **z = f(x, y) = 3*(1-x)^2*exp(-(x^2) - (y+1)^2) - 10*(x/5 - x^3 - y^5)*exp(-x^2-y^2) -1/3*exp(-(x+1)^2 - y^2).**



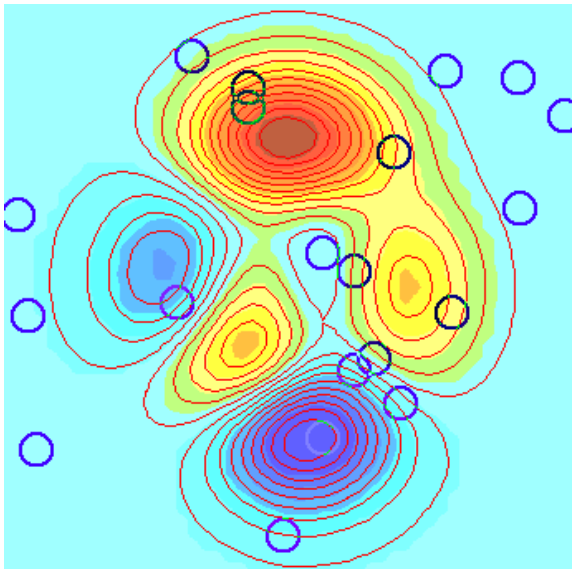


Genetic Algorithms - Summary

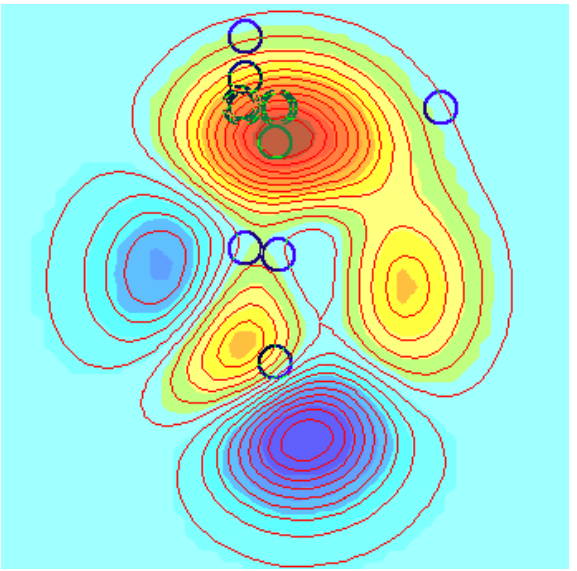
- **Derivatives of the "peaks" function**
  - **dz/dx = -6*(1-x)*exp(-x^2-(y+1)^2) - 6*(1-x)^2*x*exp(-x^2-(y+1)^2) - 10*(1/5-3*x^2)*exp(-x^2-y^2) + 20*(1/5*x-x^3-y^5)*x*exp(-x^2-y^2) - 1/3*(-2*x-2)*exp(-(x+1)^2-y^2)**
  - **dz/dy = 3*(1-x)^2*(-2*y-2)*exp(-x^2-(y+1)^2) + 50*y^4*exp(-x^2-y^2) + 20*(1/5*x-x^3-y^5)*y*exp(-x^2-y^2) + 2/3*y*exp(-(x+1)^2-y^2)**
  - **d(dz/dx)/dx = 36*x*exp(-x^2-(y+1)^2) - 18*x^2*exp(-x^2-(y+1)^2) - 24*x^3*exp(-x^2-(y+1)^2) + 12*x^4*exp(-x^2-(y+1)^2) + 72*x*exp(-x^2-y^2) - 148*x^3*exp(-x^2-y^2) - 20*y^5*exp(-x^2-y^2) + 40*x^5*exp(-x^2-y^2) + 40*x^2*exp(-x^2-y^2)*y^5 -2/3*exp(-(x+1)^2-y^2) - 4/3*exp(-(x+1)^2-y^2)*x^2 -8/3*exp(-(x+1)^2-y^2)*x**
  - **d(dz/dy)/dy = -6*(1-x)^2*exp(-x^2-(y+1)^2) + 3*(1-x)^2*(-2*y-2)^2*exp(-x^2-(y+1)^2) + 200*y^3*exp(-x^2-y^2)-200*y^5*exp(-x^2-y^2) + 20*(1/5*x-x^3-y^5)*exp(-x^2-y^2) - 40*(1/5*x-x^3-y^5)*y^2*exp(-x^2-y^2) + 2/3*exp(-(x+1)^2-y^2)-4/3*y^2*exp(-(x+1)^2-y^2)**

Genetic Algorithms - Summary

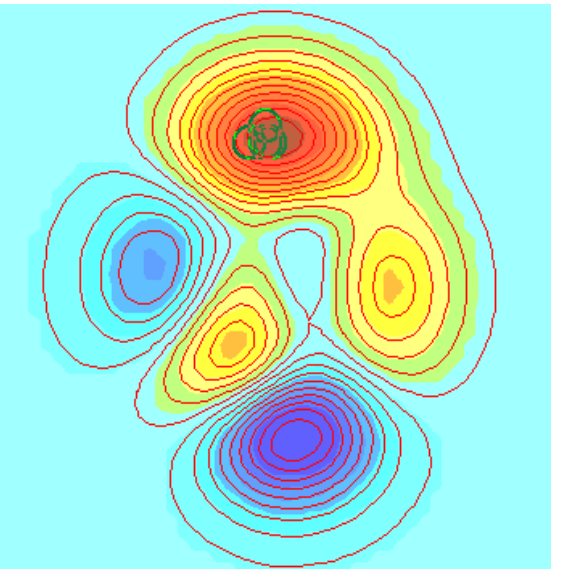
- **GA process:**



Initial population



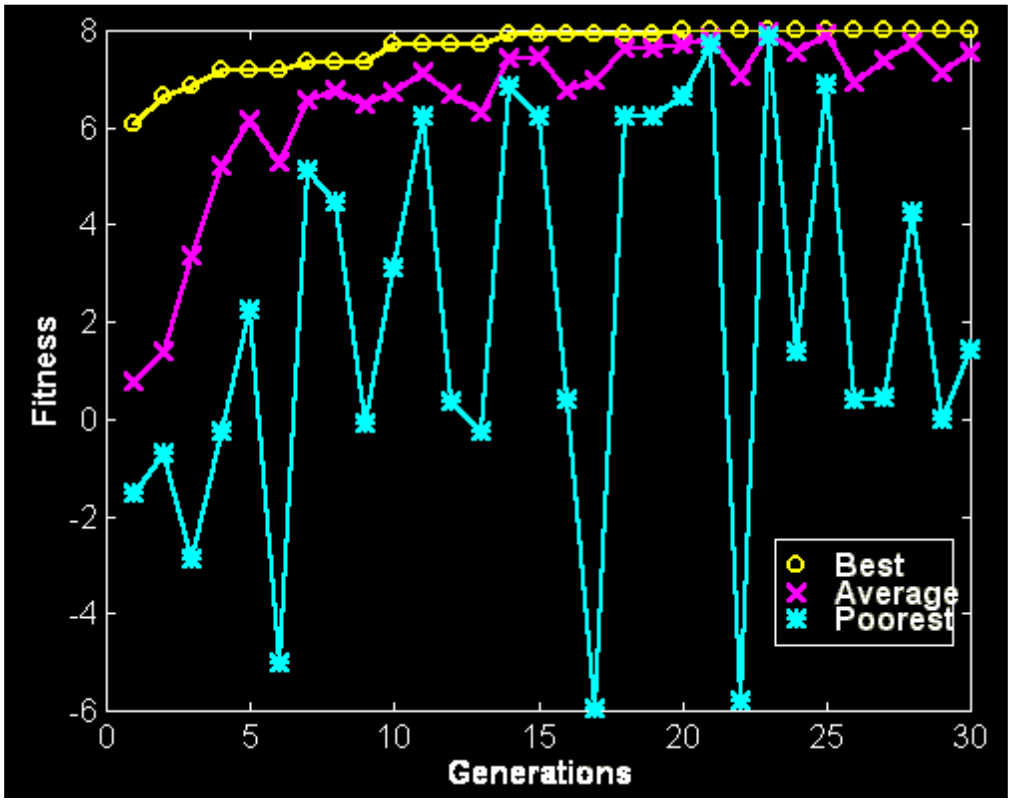
5th generation



10th generation

Genetic Algorithms - Summary

- **Performance profile**



MATLAB file: go_ga.m

Evolution Strategies

Another approach to simulating natural evolution was proposed in Germany in the early 1960s.

– Unlike genetic algorithms, this approach – called an **evolution strategy** – was designed to solve technical optimisation problems.

Evolution Strategies

- In **1963** two students of the Technical University of Berlin, **Ingo Rechenberg** and **Hans-Paul Schwefel**, were working on the search for the optimal shapes of bodies in a flow. They decided to try random changes in the parameters defining the shape following the example of natural mutation. As a result, the evolution strategy was born.
- Evolution strategies were developed as an alternative to the engineer's intuition.
- Unlike GAs, **evolution strategies use only a mutation operator**.

Basic evolution strategies

In its simplest form, termed as a (1+1)-evolution strategy, one parent generates one offspring per generation by applying *normally distributed* mutation. The (1+1)-evolution strategy can be implemented as follows:

Step 1: Choose the number of parameters N to represent the problem, and then determine a feasible range for each parameter:

$$\{x_{1min}, x_{1max}\}, \{x_{2min}, x_{2max}\}, \ldots, \{x_{Nmin}, x_{Nmax}\}$$

Define a standard deviation for each parameter and the function to be optimised.

Basic evolution strategies

Step 2: Randomly select an initial value for each parameter from the respective feasible range. The set of these parameters will constitute the initial population of parent parameters:

$$x_1, x_2, \ldots, x_N$$

Step 3: Calculate the solution associated with the parent parameters:

$$X = f(x_1, x_2, \ldots, x_N)$$

Basic evolution strategies

Step 4: Create a new (offspring) parameter by **adding a normally distributed random variable** a with **mean zero** and **pre-selected deviation** δ to **each parent parameter**:

$$x_i' = x_i + a(0,\ \delta) \quad i = 1, 2, \ldots, N$$

Normally distributed mutations with mean zero reflect the natural process of evolution (smaller changes occur more frequently than larger ones).

Step 5: Calculate the solution associated with the offspring parameters:

$$X' = f\left(x_1', x_2', \ldots, x_N'\right)$$

Basic evolution strategies

Step 6: Compare the solution associated with the offspring parameters with the one associated with th parent parameters. If the solution for the offspring is **better** than that for the parents, **replace** the parent population with the offspring population.
Otherwise, keep the parent parameters.

Step 7: Go to Step 4, and repeat the process until a satisfactory solution is reached, or a specified number of generations is considered.

Basic evolution strategies

- An evolution strategy reflects the nature of a chromosome.
- A single gene may simultaneously affect several characteristics of the living organism.
- On the other hand, a single characteristic of an individual may be determined by the simultaneous interactions of several genes.
- The natural selection acts on a collection of genes, not on a single gene in isolation.

Genetic programming

- One of the central problems in computer science is how to make computers solve problems without being explicitly programmed to do so.
- Genetic programming offers a solution through the evolution of computer programs by methods of natural selection.
- In fact, genetic programming is an extension of the conventional genetic algorithm, but the goal of genetic programming is not just to evolve a bit-string representation of some problem but the computer code that solves the problem.

Genetic programming

- Genetic programming is a recent development in the area of evolutionary computation. It was greatly stimulated in the 1990s by **John Koza**.
- According to Koza, genetic programming searches the space of possible computer programs for a program that is highly fit for solving the problem at hand.
- Any computer program is a sequence of operations (functions) applied to values (arguments), but different programming languages may include different types of statements and operations, and have different syntactic restrictions.

How do we apply genetic programming to a problem?

Before applying genetic programming to a problem, we must accomplish *five preparatory steps*:

1. Determine the set of terminals.
2. Select the set of primitive functions.
3. Define the fitness function.
4. Decide on the parameters for controlling the run.
5. Choose the method for designating a result of the run (e.g. the best-so-far generated program).

Genetic programming - example

- The Pythagorean Theorem helps us to illustrate these preparatory steps and demonstrate the potential of genetic programming. The theorem says that the hypotenuse, *c*, of a right triangle with short sides *a* and *b* is given by

$$c = \sqrt{a^2 + b^2}$$

- The aim of genetic programming is to discover a program that matches this function.

Genetic programming - example

– To measure the performance of the as-yet-undiscovered computer program, we will use a number of different **fitness cases**. The **fitness cases** for the Pythagorean Theorem are **represented by the samples of right triangles** in Table. These fitness cases are **chosen at random** over a range of values of variables *a* and *b*.

Side *a*	Side *b*	Hypotenuse *c*	Side *a*	Side *b*	Hypotenuse *c*
3	5	5.830952	12	10	15.620499
8	14	16.124515	21	6	21.840330
18	2	18.110770	7	4	8.062258
32	11	33.837849	16	24	28.844410
4	3	5.000000	2	9	9.219545

Genetic programming - example

Step 1: ***Determine the set of terminals.***

The terminals correspond to the inputs of the computer program to be discovered. Our program takes two inputs, *a* and *b*.

Step 2: ***Select the set of primitive functions.***

The functions can be presented by standard arithmetic operations, standard programming operations, standard mathematical functions, logical functions or domain-specific functions. Our program will use four standard arithmetic operations +, –, * and /, and one mathematical function *sqrt*.

Genetic programming - example

Step 3: ***Define the fitness function.*** A fitness function evaluates how well a particular computer program can solve the problem. For our problem, the fitness of the computer program can be measured by **the error between the actual result** produced by the program and the correct result given by **the fitness case**. Typically, the error is not measured over just one fitness case, but instead calculated as a sum of the absolute errors over a number of fitness cases. The closer this sum is to zero, the better the computer program.

Genetic programming - example

Step 4: ***Decide on the parameters for controlling the run.*** For controlling a run, genetic programming uses the same primary parameters as those used for GAs. They include the **population size** and the **maximum number of generations** to be run.

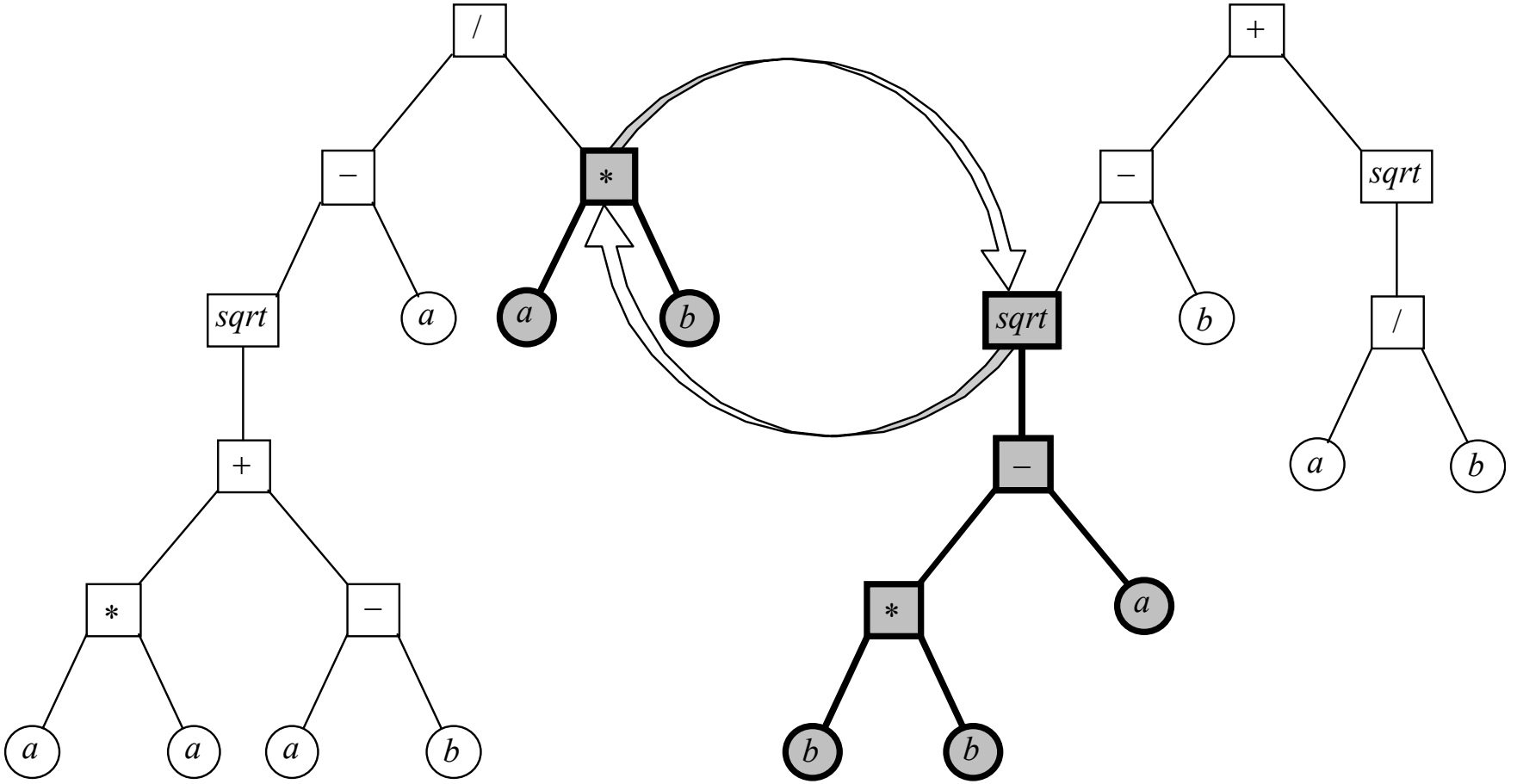
Step 5: ***Choose the method for designating a result of the run.*** It is common practice in genetic programming to designate **the best-so-far generated program** as the result of a run.

Genetic programming

Once these five steps are complete, a run can be made. The run of genetic programming **starts with a random generation** of an initial population of computer programs. Each program is composed of functions +, –, *, / and *sqrt*, and terminals *a* and *b*.

In the initial population, all computer programs usually have poor fitness, but some individuals are more fit than others. Just as a fitter chromosome is more likely to be selected for reproduction, so a fitter computer program is more likely to survive by copying itself into the next generation.

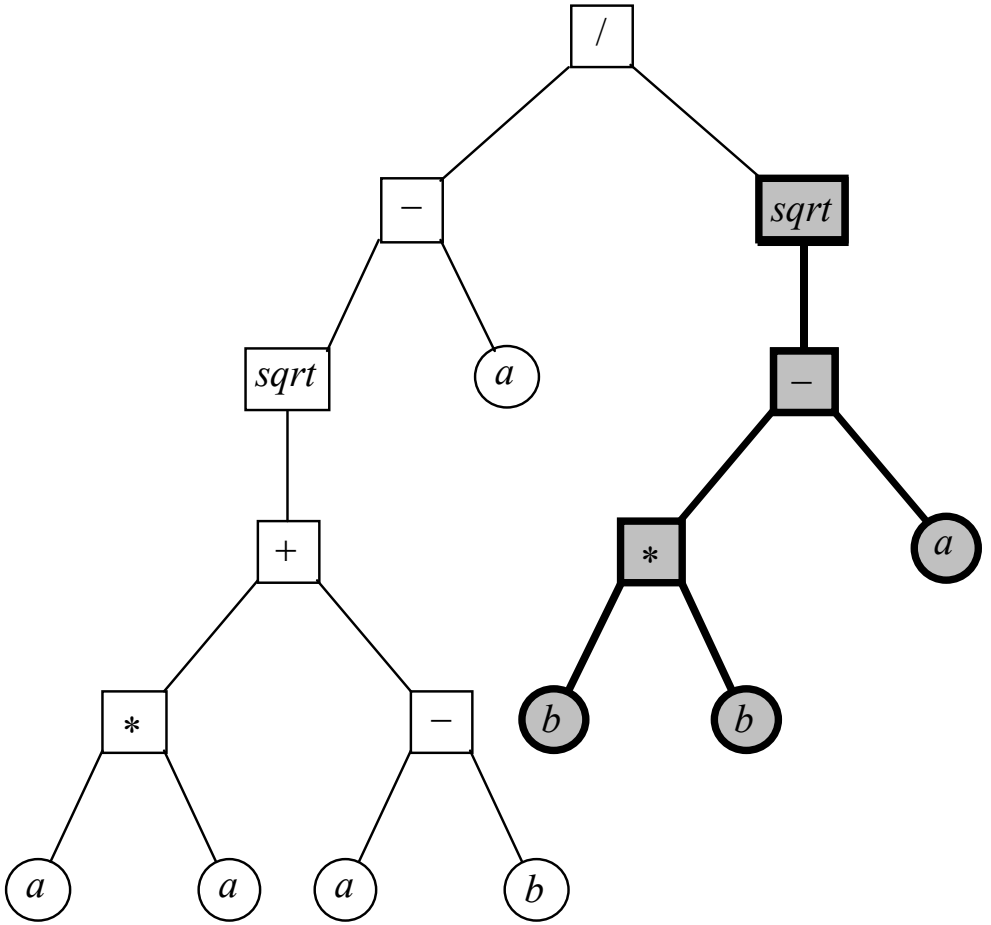
Crossover in genetic programming: Two parental expressions

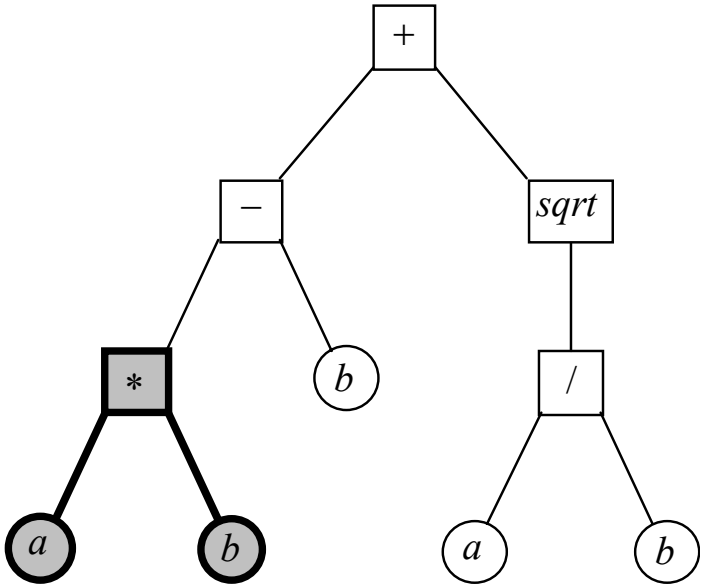


(/ (− (*sqrt* (+ (∗ *a a*) (− *a b*))) *a*) (∗ *a b*))

(+ (− (*sqrt* (− (∗ *b b*) *a*)) *b*) (*sqrt* (/ *a b*)))

Crossover in genetic programming: Two offspring expressions





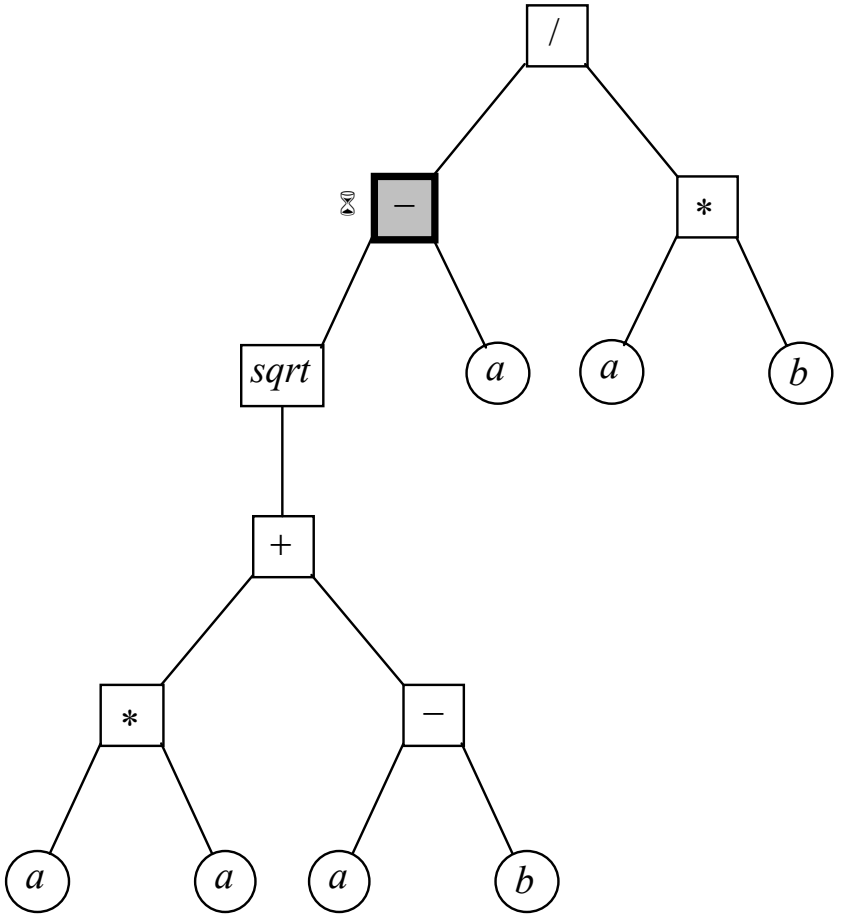
(/ (− (*sqrt* (+ (∗ *a a*) (− *a b*))) *a*) (*sqrt* (− (∗ *b b*) *a*)))

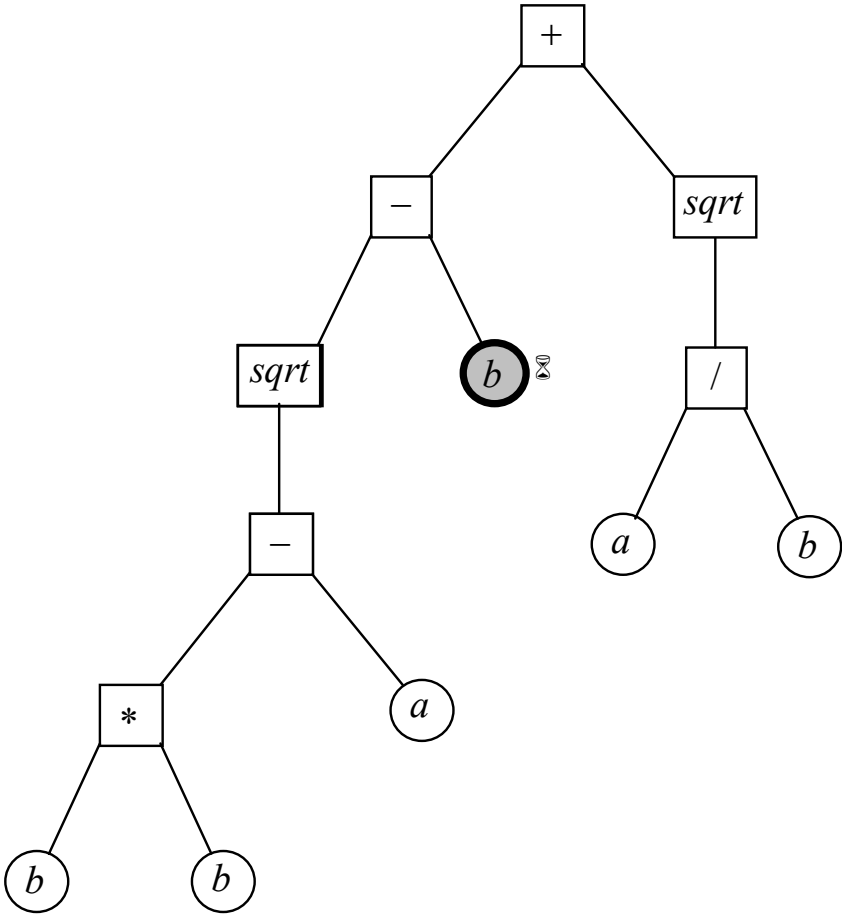
(+ (−(∗ *a b*) *b*) (*sqrt* (/ *a b*)))

Mutation in genetic programming

- A mutation operator can randomly change any function or any terminal.
- Under mutation, a function can only be replaced by a function and a terminal can only be replaced by a terminal.

Mutation in genetic programming: Original expressions

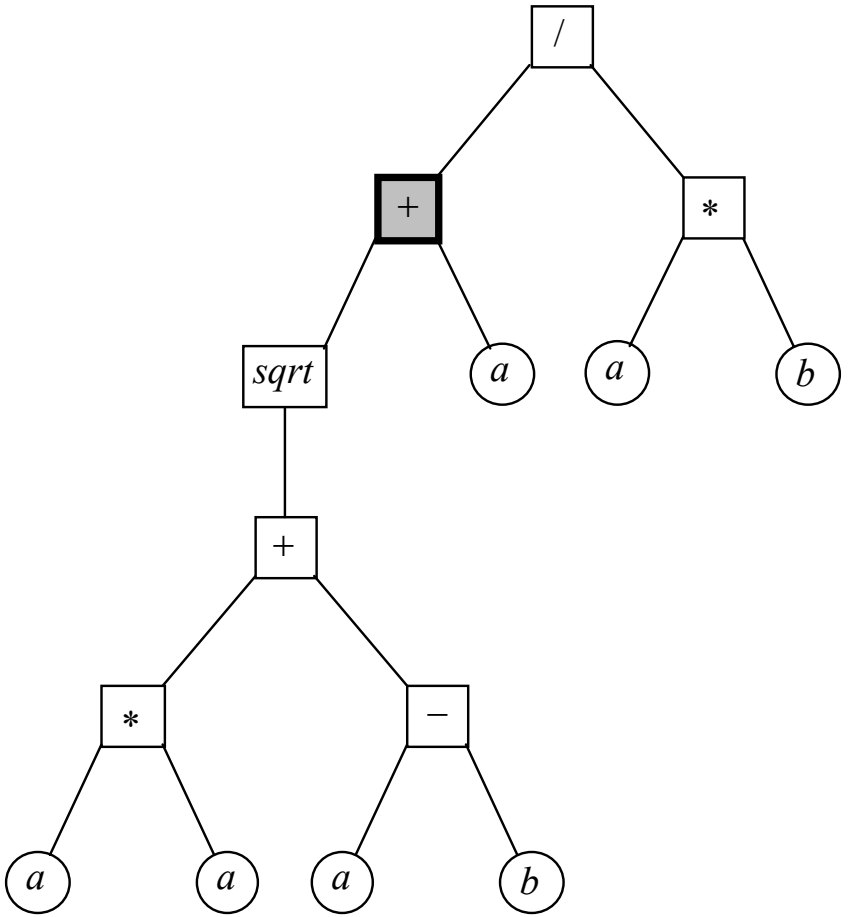




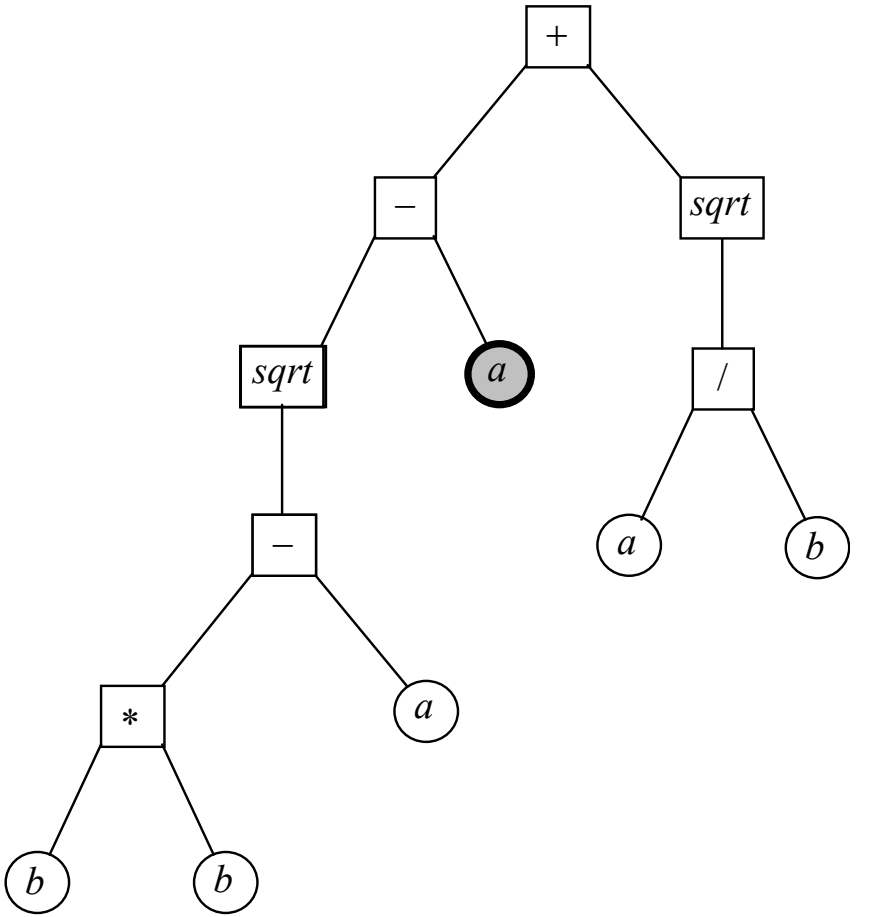
(/ (− (*sqrt* (+ (∗ *a a*) (− *a b*))) *a*) (∗ *a b*))

(+ (− (*sqrt* (− (∗ *b b*) *a*)) *b*) (*sqrt* (/ *a b*)))

Mutation in genetic programming: Mutated expressions



(/ (+ (*sqrt* (+ (∗ *a a*) (− *a b*))) *a*) (∗ *a b*))



(+ (− (*sqrt* (− (∗ *b b*) *a*)) *a*) (*sqrt* (/ *a b*)))

Genetic programming

In summary, genetic programming creates computer programs by executing the following steps:

Step 1: Assign the maximum number of generations to be run and probabilities for cloning, crossover and mutation. Note that the sum of the probability of cloning, the probability of crossover and the probability of mutation must be equal to one.

Step 2: Generate an initial population of computer programs of size *N* by combining randomly selected functions and terminals.

Genetic programming

Step 3: Execute each computer program in the population and calculate its fitness with an appropriate fitness function. Designate the best-so-far individual as the result of the run.

Step 4: With the assigned probabilities, select a genetic operator to perform cloning, crossover or mutation.

Genetic programming

Step 5: If the cloning operator is chosen, select one computer program from the current population of programs and copy it into a new population.

- If the crossover operator is chosen, select a pair of computer programs from the current population, create a pair of offspring programs and place them into the new population.
- If the mutation operator is chosen, select one computer program from the current population, perform mutation and place the mutant into the new population.

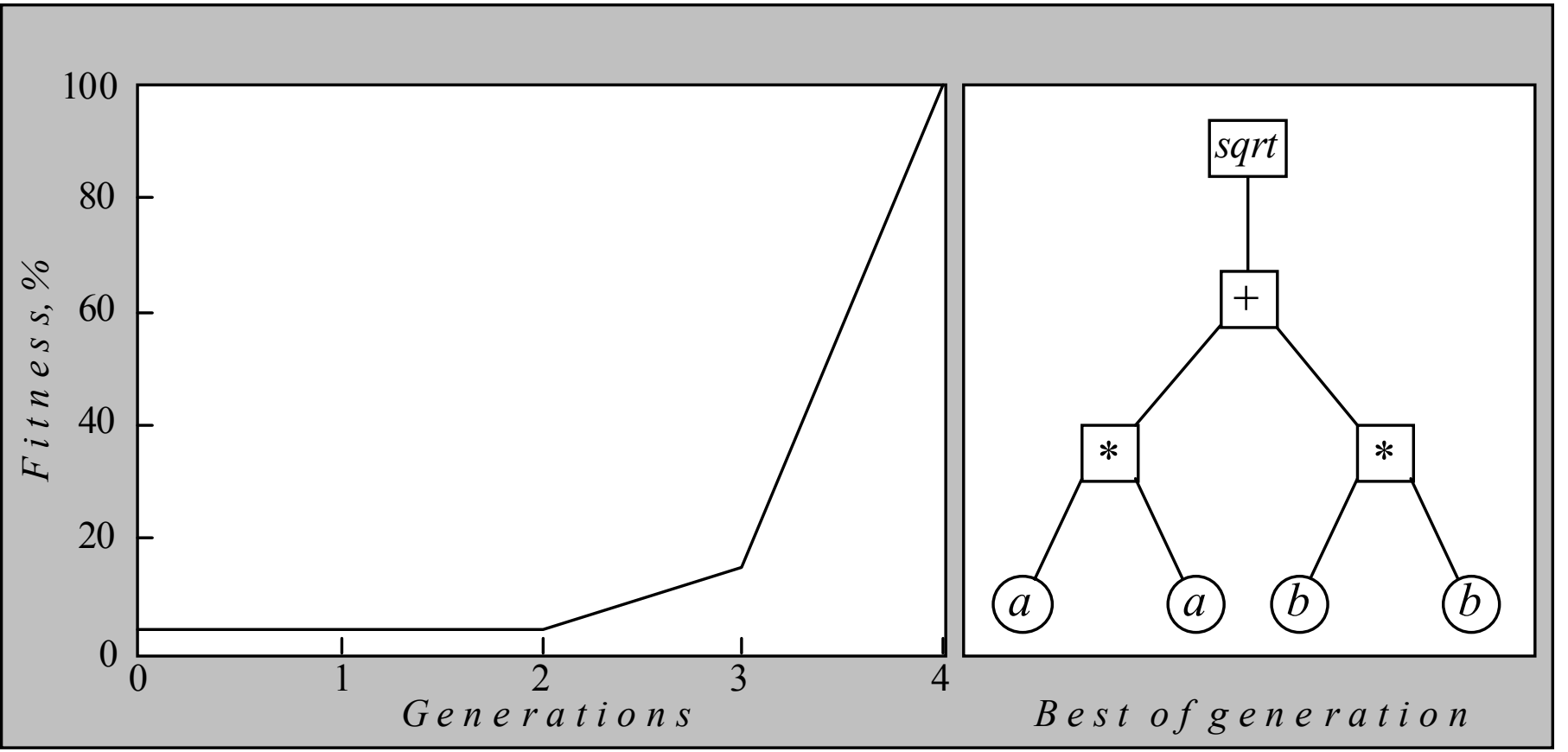
Genetic programming

Step 6: Repeat *Step 4* until the size of the new population of computer programs becomes equal to the size of the initial population, *N*.

Step 7: Replace the current (parent) population with the new (offspring) population.

Step 8: Go to *Step 3* and repeat the process until the termination criterion is satisfied.

Fitness history of the best expression



What are the main advantages of genetic programming compared to genetic algorithms?

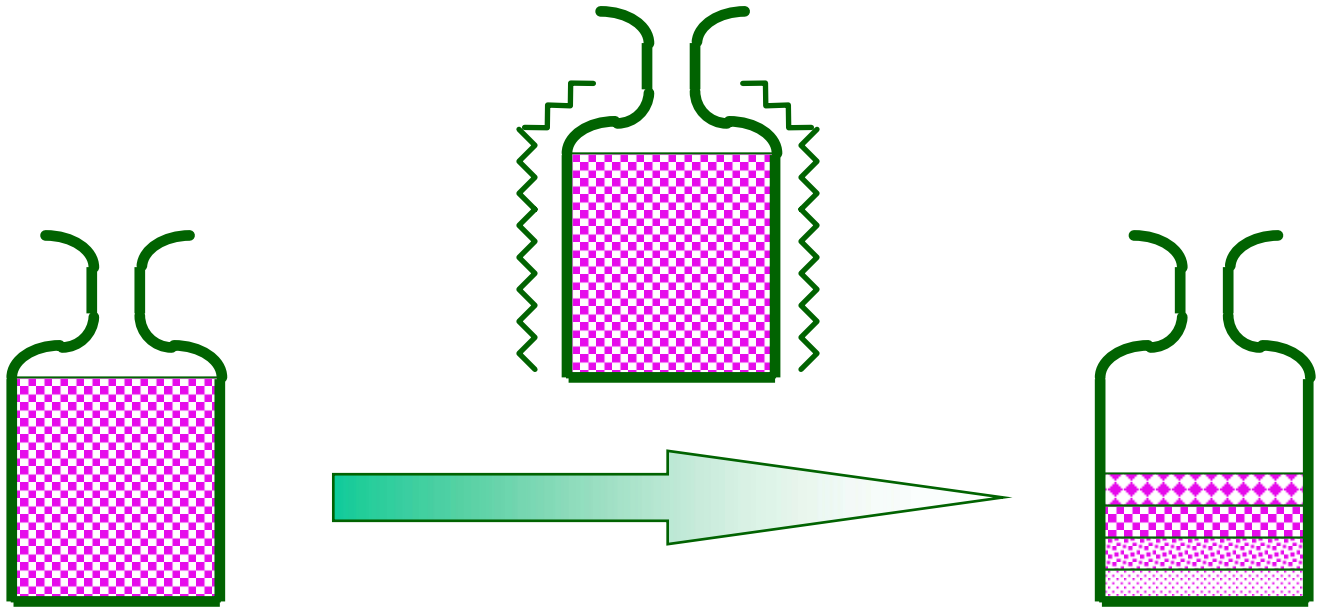
- Genetic programming applies the same evolutionary approach. However, genetic programming is no longer breeding bit strings that represent coded solutions but complete computer programs that solve a particular problem.
- The **fundamental difficulty** of GAs lies in the **problem representation**, that is, in the **fixed-length coding**. A poor representation limits the power of a GA, and even worse, may lead to a false solution.

Genetic programming

- A fixed-length coding is rather artificial. As it cannot provide a dynamic variability in length, such a coding often causes considerable redundancy and reduces the efficiency of genetic search.
- In contrast, genetic programming uses high-level building blocks of variable length. Their size and complexity can change during breeding.
- Genetic programming works well in a large number of different cases and has many potential applications.

Other Derivative-free Methods: Simulated Annealing

- **Analogy**



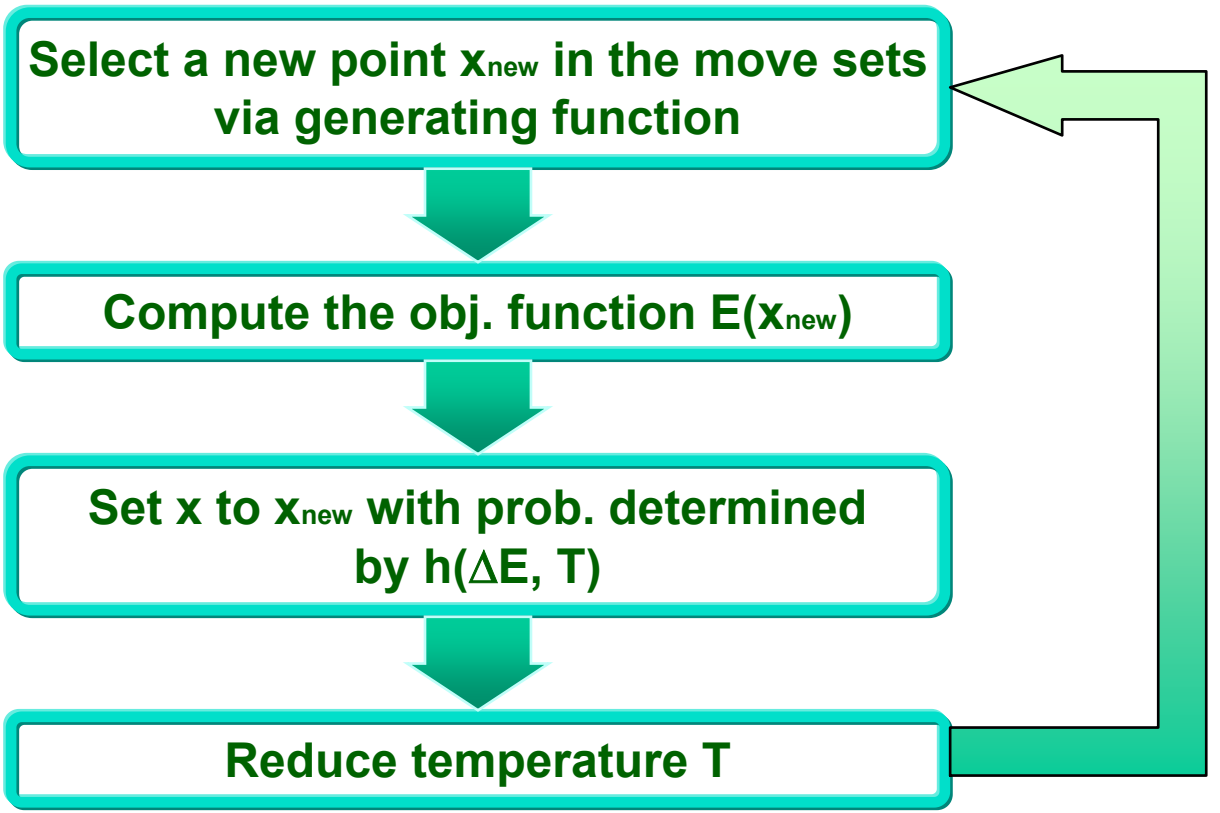
Simulated Annealing

Terminology:

- Objective function E=f(x):
  function to be optimized
- Move set: set of next points to explore
- Generating function g(Δx,T):
  to select next point $\Delta x = x_{new} - x$
- Acceptance function h(ΔE,T):
  to determine if the selected point should be accept or not.
  Usually $h(\Delta E, T) = 1/(1+\exp(\Delta E/(cT))$.
  $\Delta E = f(x_{new}) - f(x)$ and c is a system dependent constant
- Annealing (cooling) schedule:
  schedule for reducing the temperature T

Simulated Annealing

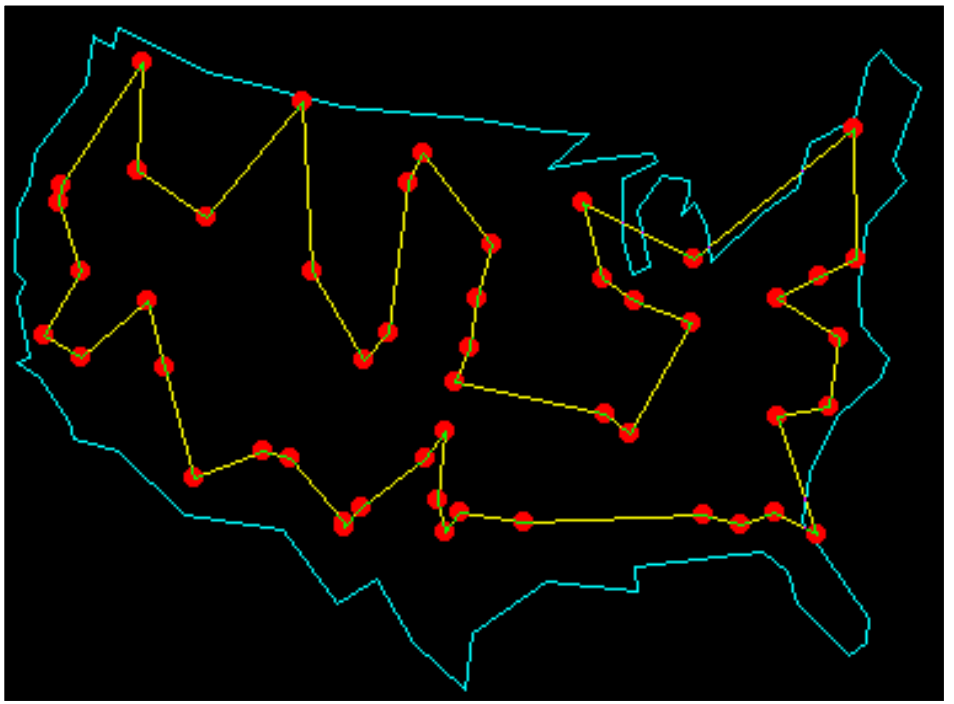
- **Flowchart:**



Simulated Annealing

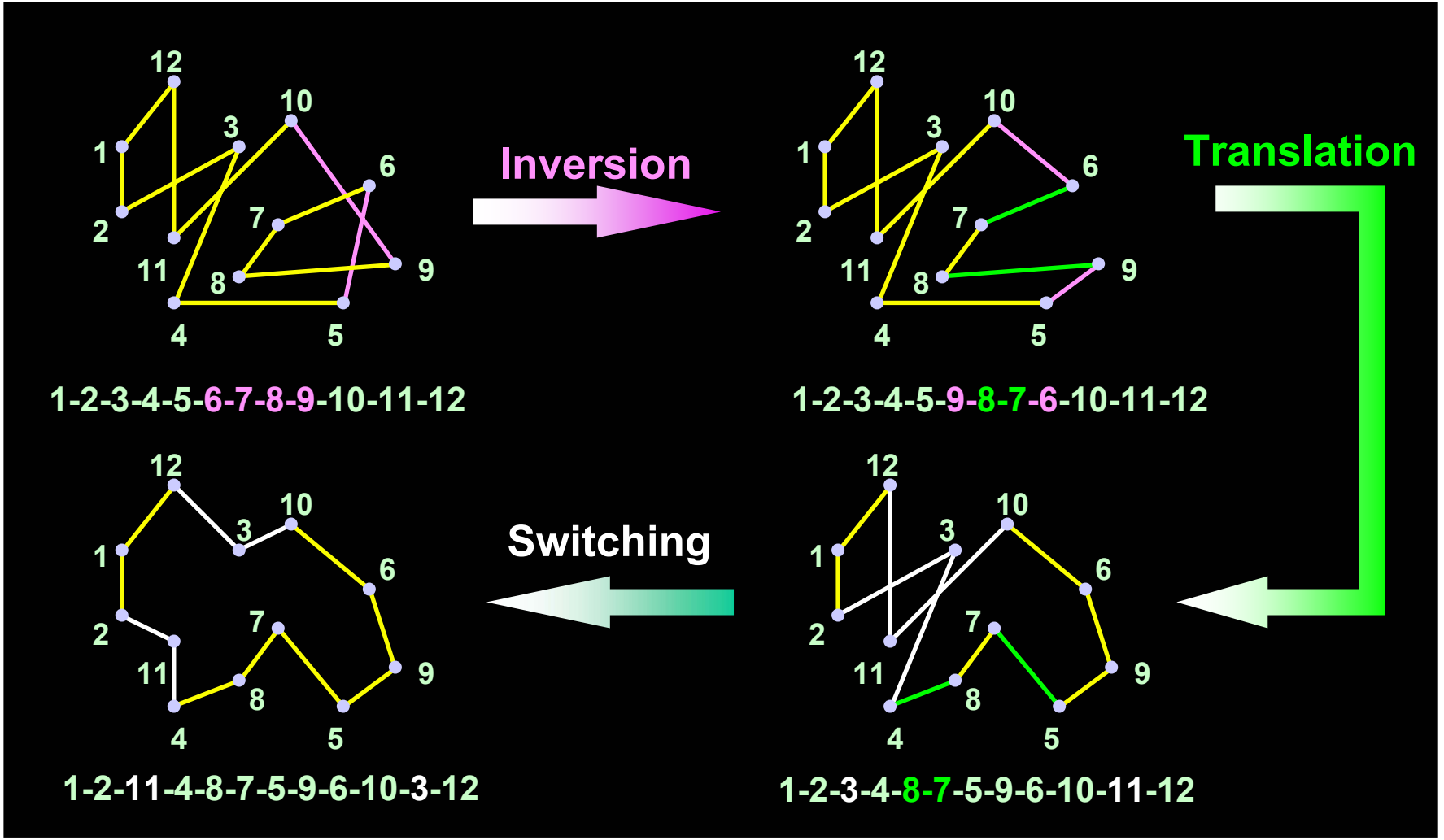
- **Example: Travel Salesperson Problem (TSP)**

How to transverse n cities once and only once with a minimal total distance?



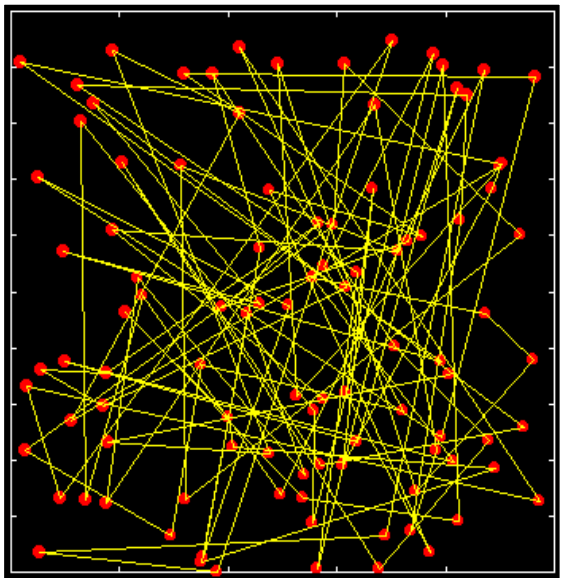
Simulated Annealing

- **Move sets for TSP**

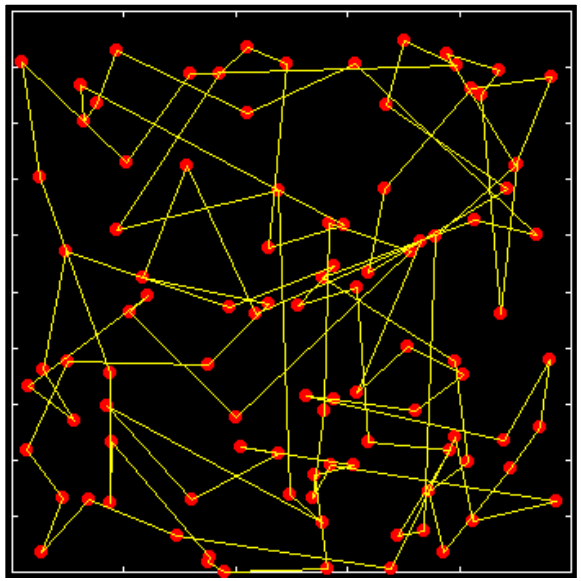


Simulated Annealing

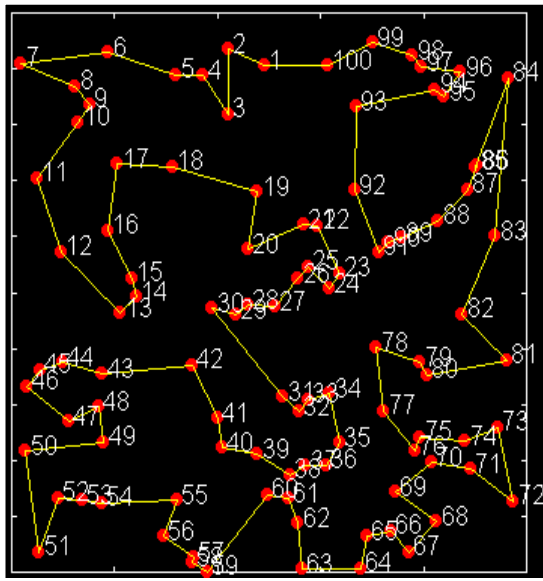
- **A 100-city TSP using SA**



Initial random path



During SA process



Final path

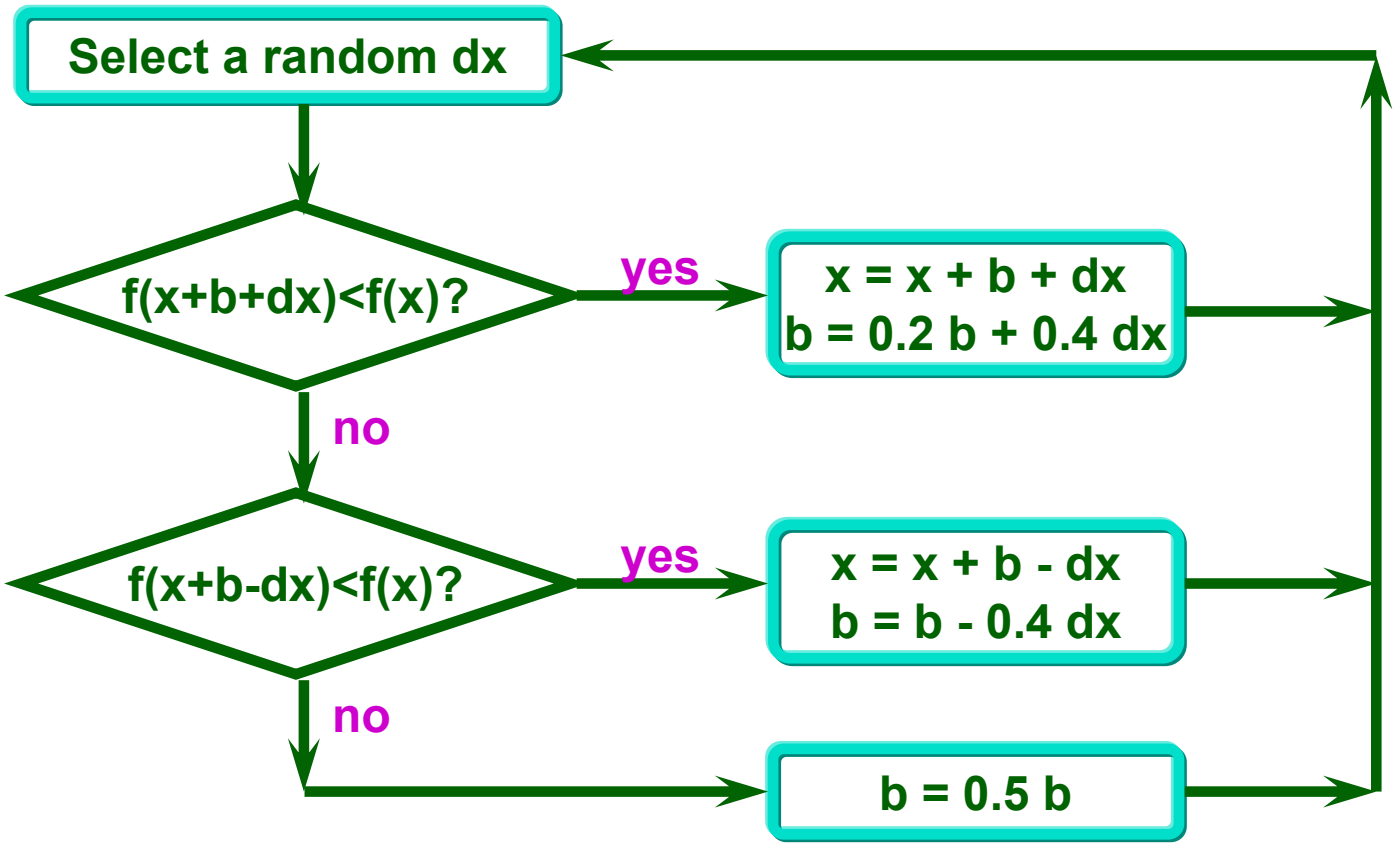
Other Derivative-free Methods: Random Search

- **Properties:**
  - **Intuitive**
  - **Simple**
- **Analogy:**
  - **Get down to a valley blindfolded**
- **Two heuristics:**
  - **Reverse step**
  - **Bias direction**

Random Search

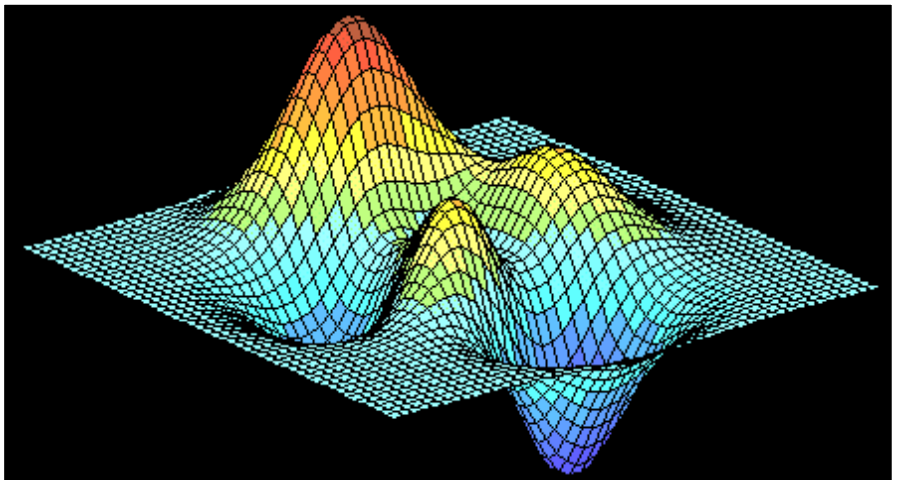
- **Flowchart:**

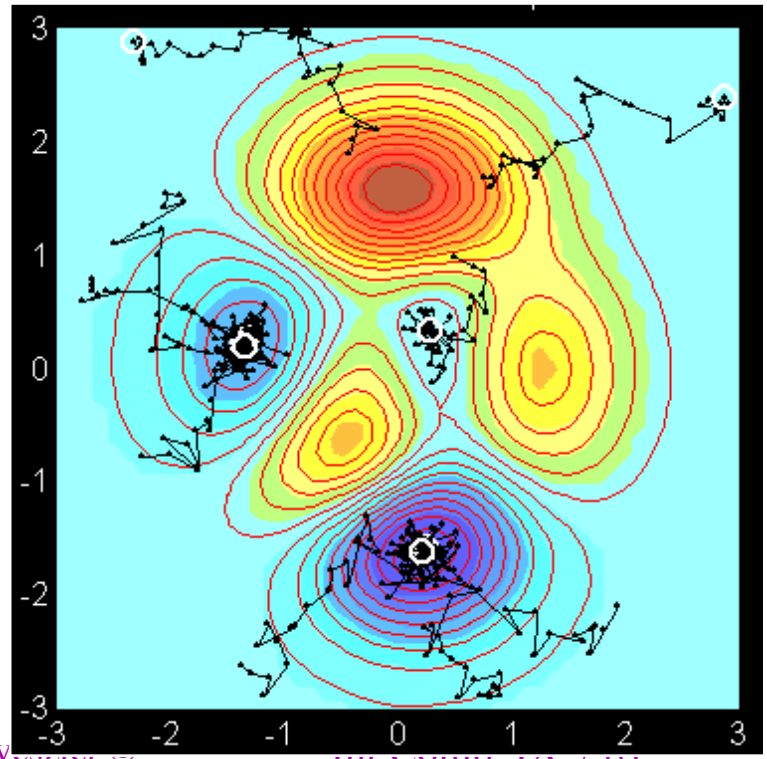
b: bias
dx: random step



Random Search

- **Example: Find the min. of the "peaks" function**
- **z = f(x, y) = 3*(1-x)^2*exp(-(x^2) - (y+1)^2) - 10*(x/5 - x^3 - y^5)*exp(-x^2-y^2) -1/3*exp(-(x+1)^2 - y^2).**





MATLAB file: go_rand.m

Ajánlott irodalom

- **The slides of this lecture are partially based on the books:**

Kóczy T. László és Tikk Domonkos: ***Fuzzy rendszerek*****,** Typotex Kiadó, 2000, **ISBN 963-9132-55-1**

J.-S. R. Jang, C.-T. Sun, E. Mizutani: ***Neuro-Fuzzy and Soft Computing*****,** Prentice Hall, 1997, **ISBN 0-13-261066-3**

Michael Negnevitsky: ***Artificial Intelligence: A Guide to Intelligent Systems*****,** Addison Wesley, Pearson Education Limited, 2002, **ISBN 0201-71159-1**