UNIVERSITY OF LOUISVILLE
J.B. SPEED SCHOOL OF ENGINEERING

Computer Engineering and Computer Science Department

# CECS545-Artificial Intelligence

## Beyond Classical Search 2

Dr. Roman Yampolskiy

# Evolution Strategies (or non-sexual evolution)

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.

- 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.

**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:

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

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

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

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:

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

**Step 6:** Compare the solution associated with the offspring parameters with the one associated with the 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.

- 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 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 algorithms 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.

- Since genetic programming manipulates programs by applying genetic operators, a programming language should permit a computer program to be manipulated as data and the newly created data to be executed as a program. For these reasons, **LISP** was chosen as the main language for genetic programming.

# LISP structure

LISP has a highly symbol-oriented structure. Its basic data structures are **atoms** and **lists**. An atom is the smallest indivisible element of the LISP syntax. The number *21*, the symbol *X* and the string *"This is a string"* are examples of LISP atoms. A list is an object composed of atoms and/or other lists. LISP lists are written as an ordered collection of items inside a pair of parentheses.

# How do we apply genetic programming to a problem?

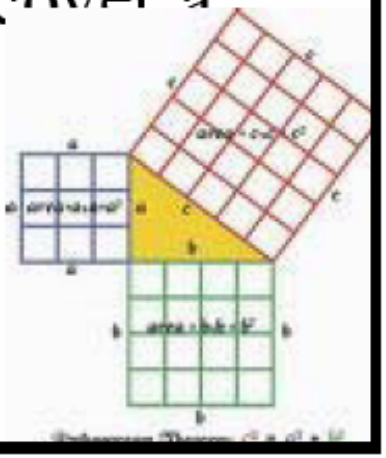
Before applying genetic programming to a problem, we must accomplish *five preparatory steps*:

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.

- 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.



- 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 |

**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*.

**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.

**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.

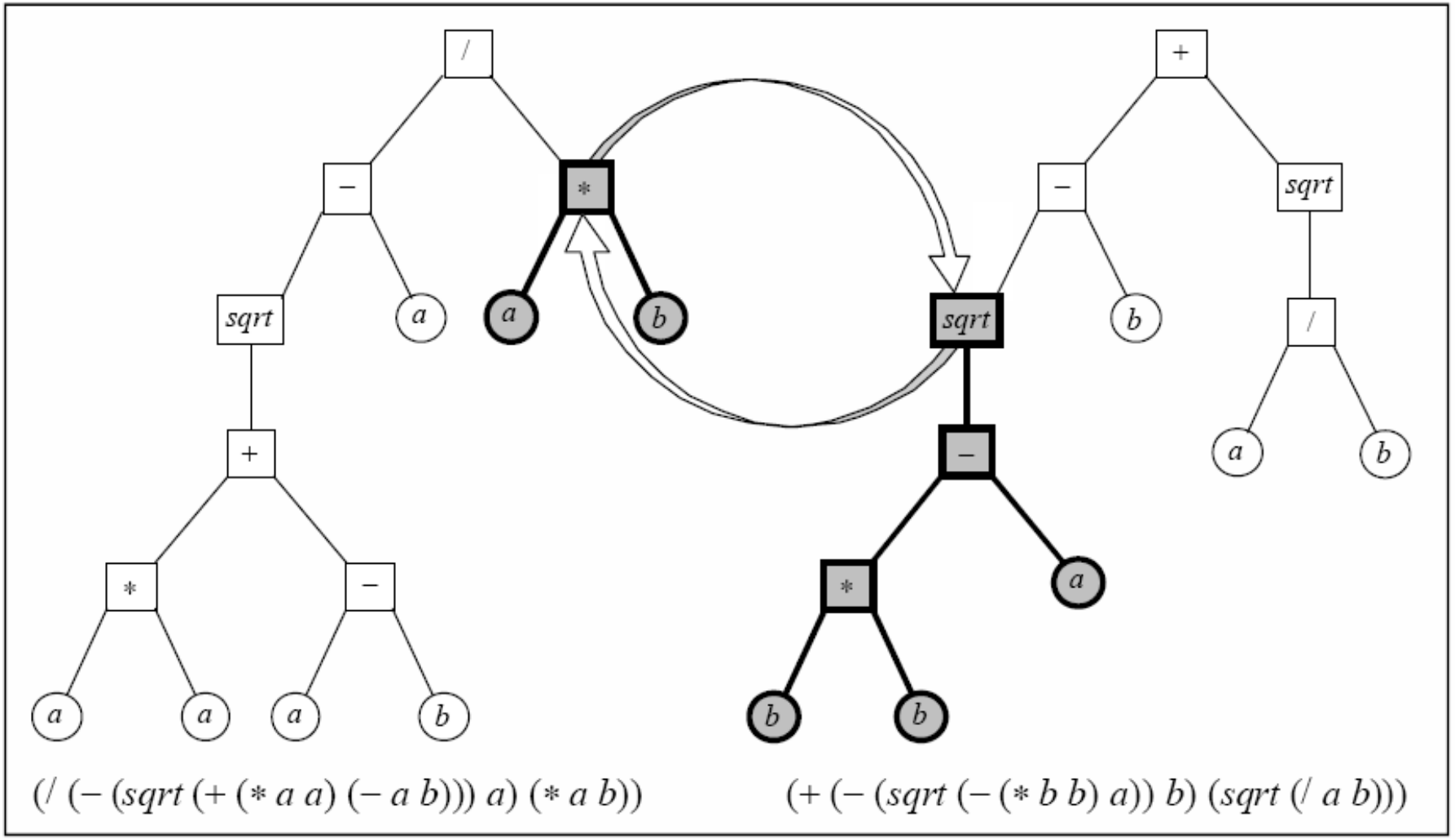
**Step 5**: ***Choose the method for designating a result of the run.*** It is common practice in genetic programming to designate the best-so-far generated program as the result of a run.

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 S-expressions*

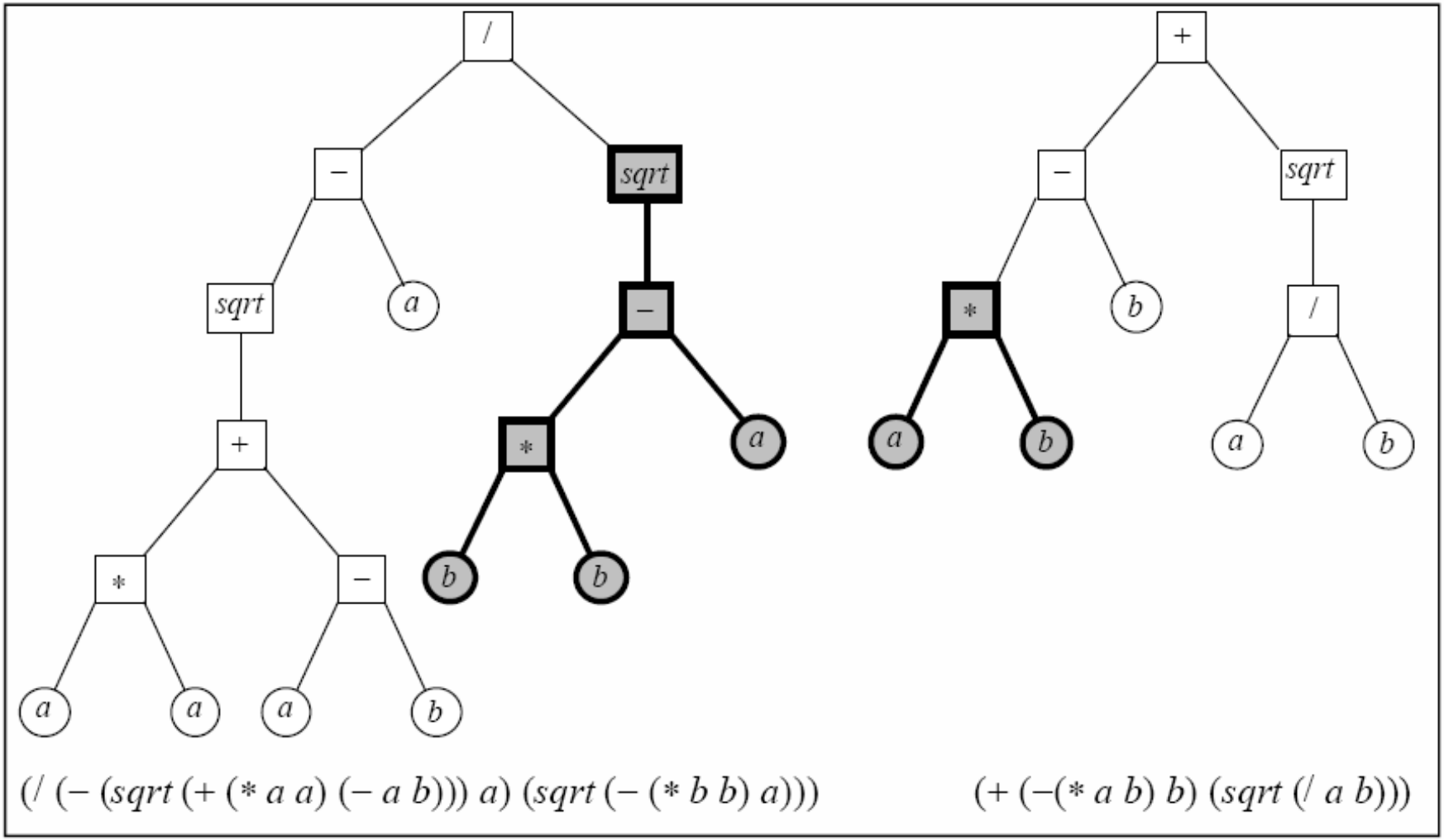



(/ (− (*sqrt* (+ (∗ *a a*) (− *a b*))) *a*) (∗ *a b*))

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

# Crossover in genetic programming:

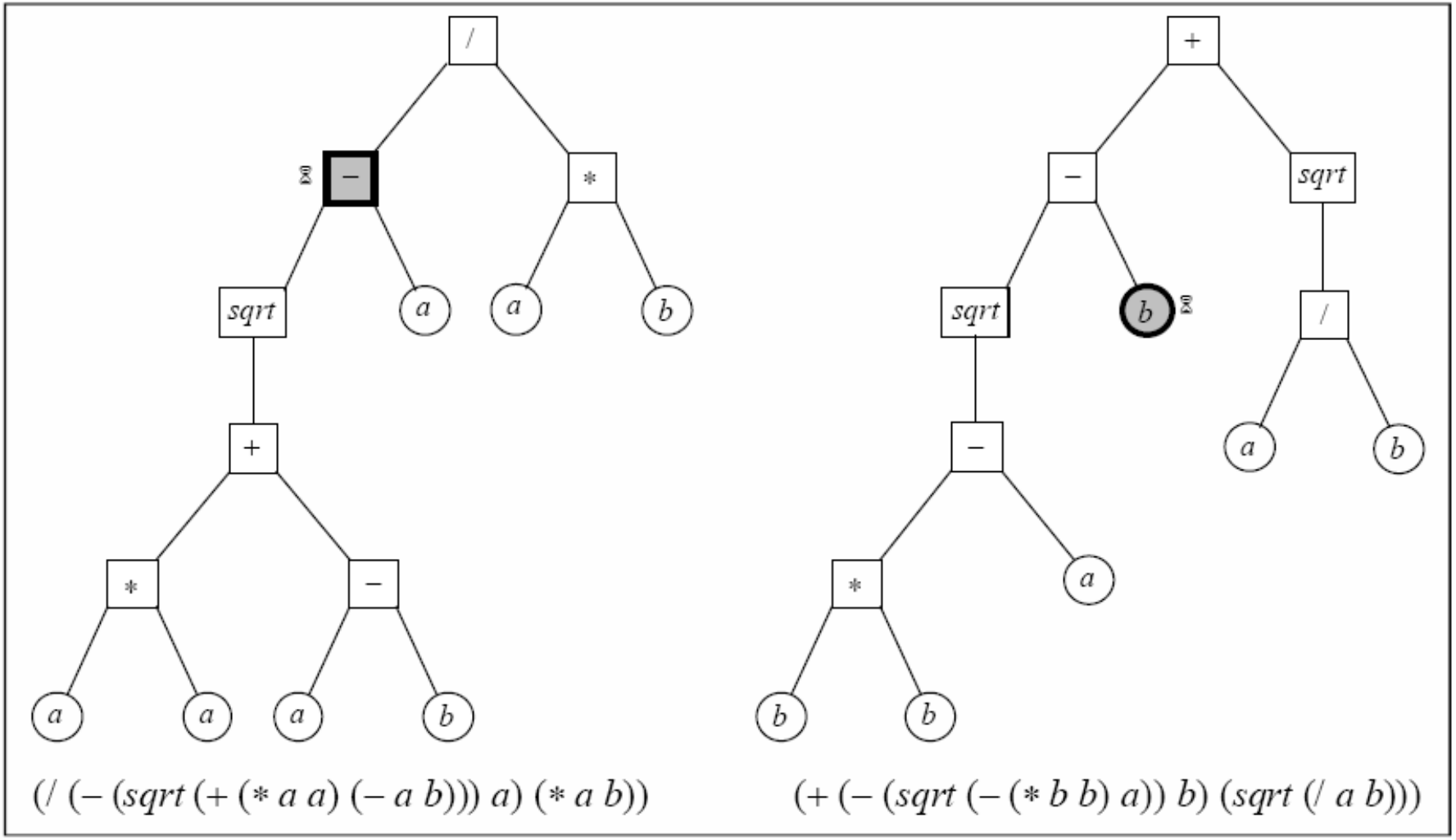
## *Two offspring S-expressions*



# Mutation in genetic programming

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

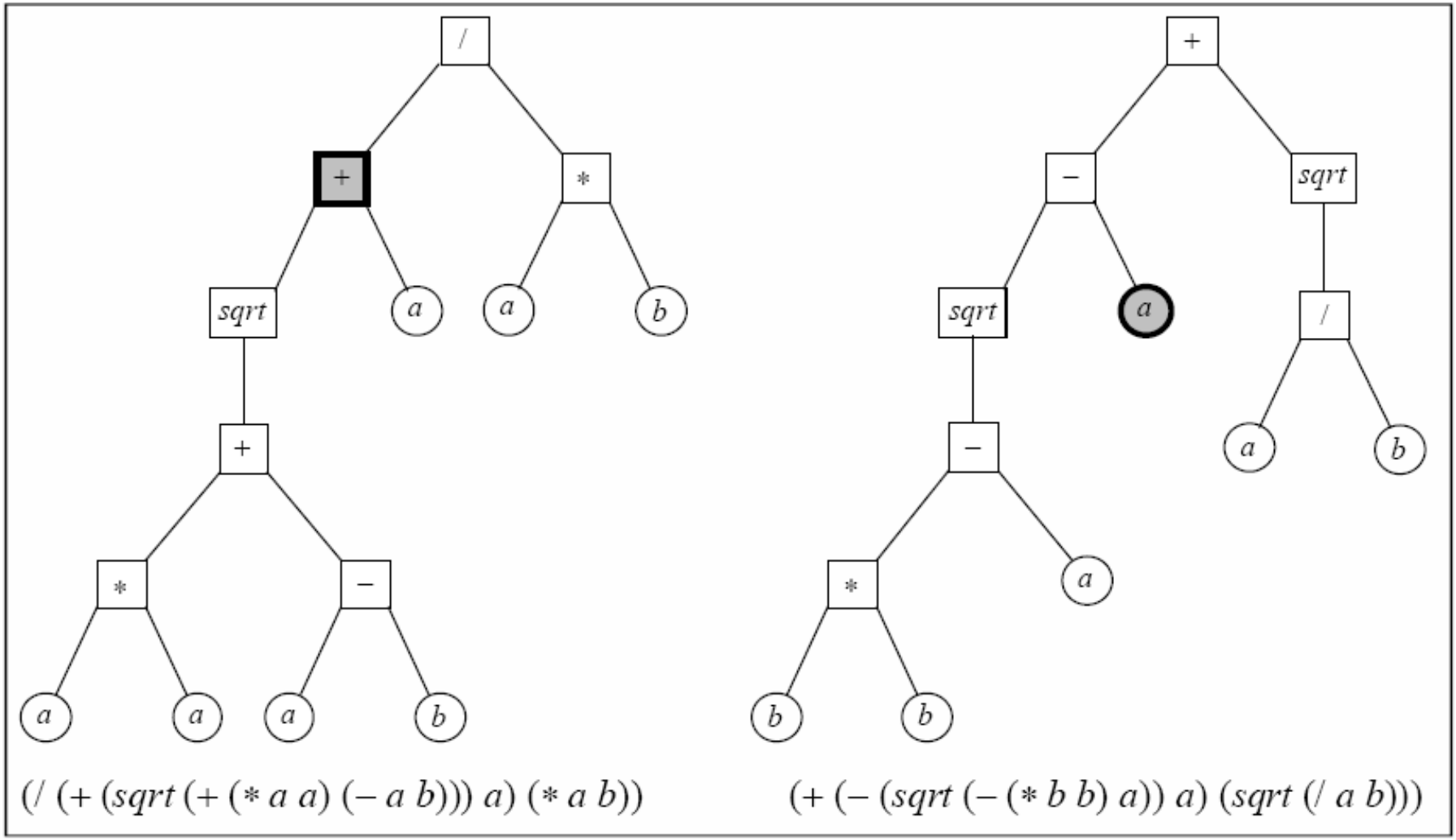
# Mutation in genetic programming:
## *Original S-expressions*




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

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

# Mutation in genetic programming:
## *Mutated S-expressions*



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.

**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.

**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.

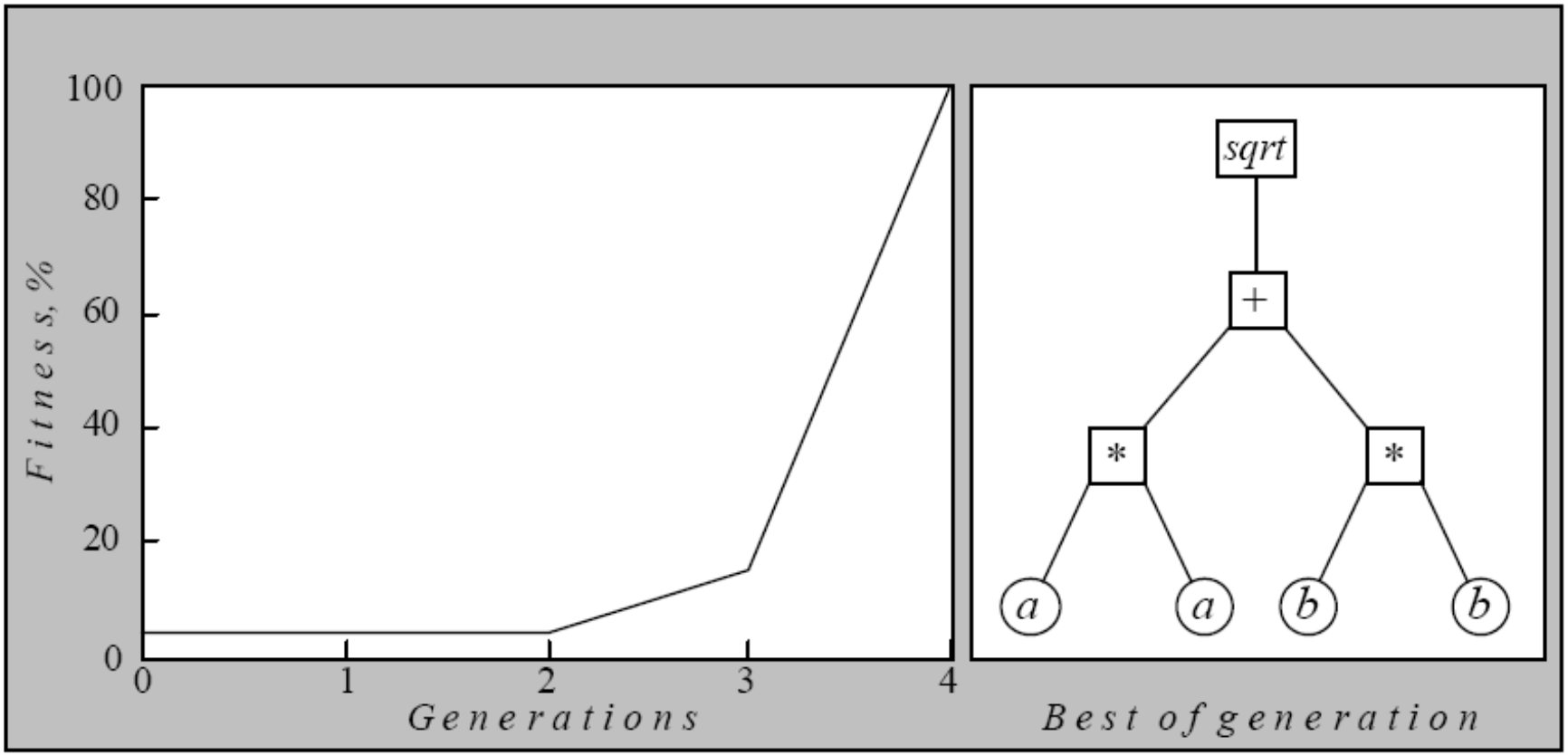
- If the crossover operator is chosen, select a pair of computer programs from the current population, create a pair of offspring programs and place them into the new population.
- If the mutation operator is chosen, select one computer program from the current population, perform mutation and place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

**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 S-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.

- 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.

# Exercises

- Design three crossover operators for the TSP problem.
- What makes a good crossover operator?
- Design a mutation operator.
- Be prepared to explain your designs.

# The End!