ISSN 1691-5402
ISBN 978-9984-44-028-6
Environment. Technology. Resources
Proceedings of the 7th International Scientific and Practical Conference. Volume I1

EVOLUTIONARY ALGORITHMS AT CHOICE: FROM GA TO GP

EVOLŪCIJAS ALGORITMI PĒC IZVĒLES: NO GA UZ GP

Peter Grabusts
Rezekne Higher Educational Institution
Atbrivoshanas al. 90, Rezekne LV 4600, Latvia
Phone: +(371)4623798, e-mail: peter@ru.lv

Abstract. *Nowadays the possibilities of evolutionary algorithms are widely used in many optimization and classification tasks. Evolutionary algorithms are stochastic search methods that try to emulate Darwin's principle of natural evolution. There are (at least) four paradigms in the world of evolutionary algorithms: evolutionary programming, evolution strategies, genetic algorithms and genetic programming. This paper analyzes present-day approaches of genetic algorithms and genetic programming and examines the possibilities of genetic programming that will be used in further research. The paper presents implementation examples that show the working principles of evolutionary algorithms.*

Keywords: *evolutionary algorithms, genetic algorithm, genetic programming.*

Evolutionary algorithms: an introduction

Evolutionary algorithms (EA) are population-based meta-heuristic optimization algorithms that use biology-inspired mechanisms like mutation, crossover, natural selection, and survival of the fittest in order to refine a set of solution candidates iteratively [1].

All EA have three features in common:

1. They use a *population* of potential solutions to the problem that is to be solved.
2. They rate the quality of these solutions with an objective function, and base the *selection* of surviving solutions on this quality measure.
3. They have a *reproduction* stage in which new solutions are constructed inheriting traits from current solutions.

Three basic mechanisms drive natural evolution: *reproduction, mutation* and *selection*. These mechanisms act on the chromosomes containing the genetic information of the individual (the genotype), rather than on the individual (the phenotype). Reproduction is the process in which new individuals are introduced into population. During reproduction, recombination or crossover occurs, transmitting to the offspring chromosomes that are common of both parent's genetic information. Mutation introduces small changes into the inherited chromosomes. Selection is a process guided by the Darwinian principle of survival of the fittest. The fittest individuals are those who are best adapted to their environment, and who thus survive and reproduce.

In EA the term *chromosome* typically refers to a candidate solution to a problem, often encoded as a bit string. The "genes" are either single bits or short blocks of adjacent bits that encode a particular element of the candidate solution (e.g., in the context of multi-parameter function optimization the bits encoding a particular parameter might be considered to be a gene). An allele in a bit string is either 0 or 1; for larger alphabets more alleles are possible at each locus. Crossover typically consists of exchanging genetic material between two single chromosome parents. Mutation consists of flipping the bit at a randomly chosen locus (or, for larger alphabets, replacing the symbol at a randomly chosen locus with a randomly chosen new symbol) [2].

In that way EA are search methods that take their inspiration from natural selection and survival of the fittest in the biological world. EAs differ from more traditional optimization techniques in the way that they involve a search from a "population" of solutions, not from a single point.

Each iteration of an EA involves a competitive selection that weeds out poor solutions. The solutions with high "fitness" are "recombined" with other solutions by swapping parts of a solution with another. Solutions are also "mutated" by making a small change to a single element of the solution. Recombination and mutation are used to generate new solutions that are biased towards regions of the space for which good solutions have already been seen.
The basic cycle of EA is shown in Figure 1 [3].

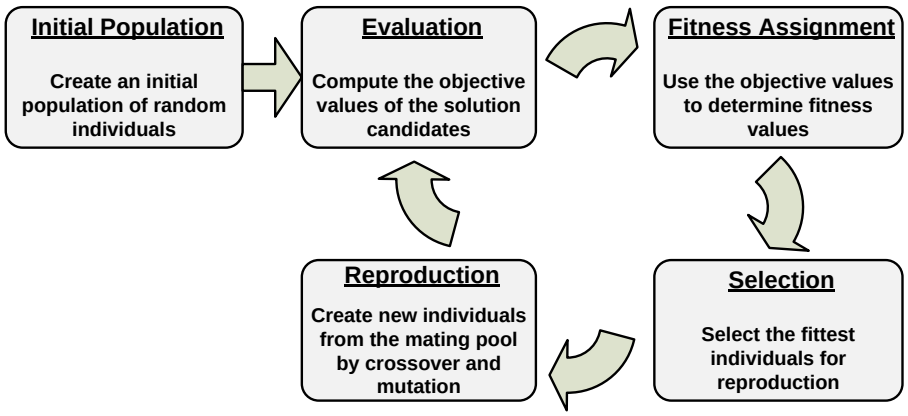


Fig. 1. **The basic cycle of EA**

There are several different types of EA [1]. These include:

- evolutionary programming (EP), which focus on optimizing continuous functions without recombination;
- evolutionary strategies (ES), which focus on optimizing continuous functions with recombination;
- genetic algorithms (GA), which focus on optimizing general combinatorial problems;
- genetic programming (GP), which evolves programs.

Genetic algorithms overview

GA is a subclass of evolutionary algorithms where the elements of the search space are binary strings or arrays of other elementary types. GA is an optimization and search technique based on the principles of genetics and natural selection. GA allows a population composed of many individuals to evolve under specified selection rules to a state that maximizes the "fitness" (i.e., minimizes the cost function) [1; 2; 6; 7]. The standard GA is as follows:

```
{  % Generate random population of chromosomes
   Initialize population;
   % Evaluate the fitness of each chromosome in the population
   Evaluate population;                                    [Fitness]
   % Create, accept, and test a new population:
   while Termination_Criteria_Not_Satisfied
   {   % Select according to fitness
       Select parents for reproduction;                    [Selection]
       % With a crossover probability perform crossover or copy parents
       Perform crossover;                                  [Crossover]
       % With a mutation probability mutate offspring at each position in chromosome
       Perform mutation;                                   [Mutation]
       Accept new generation;
       Evaluate population;                                [Fitness]
   }
}
```

The simple GA begins by defining the optimization variables, the cost function and the cost. It ends like other optimization algorithms by testing for convergence (see Figure 2) [5].

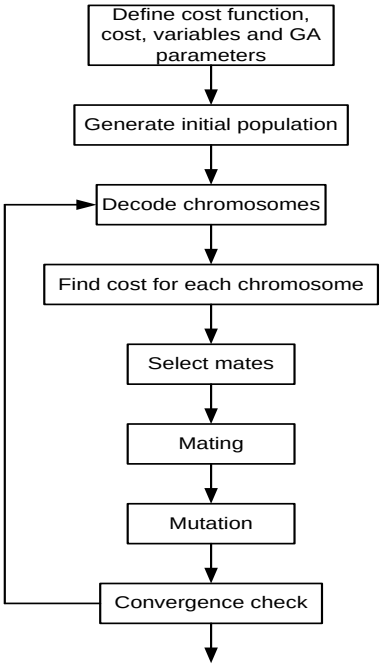


Fig. 2. **Flowchart of GA**

There are four main differences that separate GA from traditional search and optimization procedures:

1. GA uses an encoding of the parameters - not the parameters themselves;
2. GA searches from a population of search points - not a single point;
3. GA uses only the objective function to test solution quality - not other additional knowledge;
4. GA use probabilistic transition rules - not deterministic rules.

For practical purposes GA activity is implemented in the following way [6]. According to flowchart in Figure 3, the GA begins by defining a chromosome or an array of variable values to be optimized. Chromosome is written as an N_{var} element row vector – $chromosome=[p_1,p_2,...p_N]$. Each chromosome has a cost found by evaluating the cost function f at $p_1,p_2,...p_N$: $cost=f(chromosome)=f(p_1,p_2,...p_N)$. The initial population has N_{pop} chromosomes and is an N_{pop} x N_{bits} matrix filled with random ones and zeros. The GA works with the binary encodings, but the cost function requires continuous variables. Whenever the cost function is evaluated, the chromosome must first be decoded.

Next, the variables are passed to the cost function for evaluation. The N_{pop} costs and associated chromosomes are ranked from lowest cost to highest cost. Then, only the best are selected to continue, while the rest are deleted. The selection rate X_{rate} is the fraction of N_{pop} that survives for the next step of mating. The number of chromosomes that are kept in each generation is $N_{keep}=X_{rate}N_{pop}$. Natural selection occurs within each iteration of the algorithm.

Two chromosomes are selected from the mating pool of N_{keep} chromosomes to produce two new offspring. Pairing takes place in the mating population until N_{pop}-N_{keep} offspring are born to replace the discarded chromosomes.

Mating is the creation of one or more offspring from the parents selected in the pairing process. The most common form of mating involves two parents that produce two offspring (see Figure 3) [6].

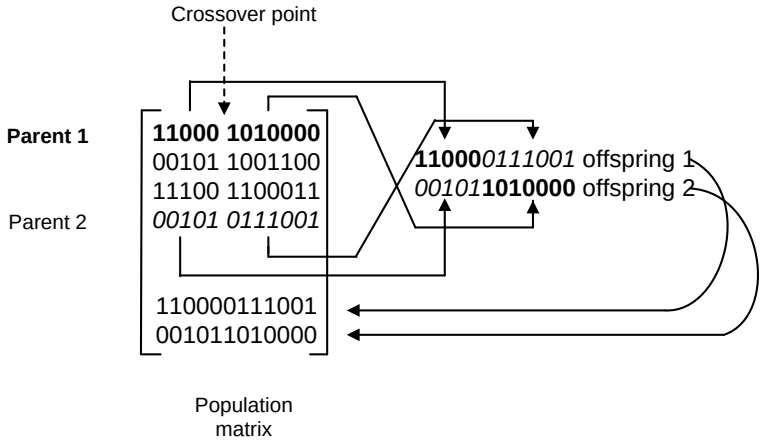


Fig. 3. **The creation of offspring**

Random mutations alter a certain percentage of the bits from the list of chromosomes. Mutation can introduce characteristics not in the original population. A single point mutation changes a 1 to a 0, and conversely. Mutation points are randomly selected from the N_{pop} *x* N_{bits} total number of bits in the population matrix. Increasing the number of mutations tends to distract the algorithm from converging on a best solution.

After the mutations take place, the costs associated with the offspring and mutated chromosomes are calculated. The number of generations that evolve depends on whether an acceptable solution is reached or a set number of iterations is exceeded. After a while, all the chromosomes and associated costs would become the same if it were not for mutations. At this point the algorithm should be stopped [6].

To illustrate the work of GA the following function has been selected **f(x)=x^2*sin(x)** with minimum: f(-8.0962)=-63.635 for -10≤x≤10 (see Fig. 4). Best cost=-63.635 and best solution=-8.0962.

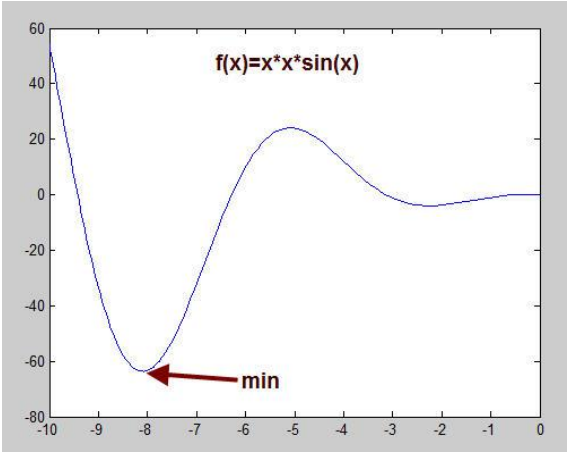


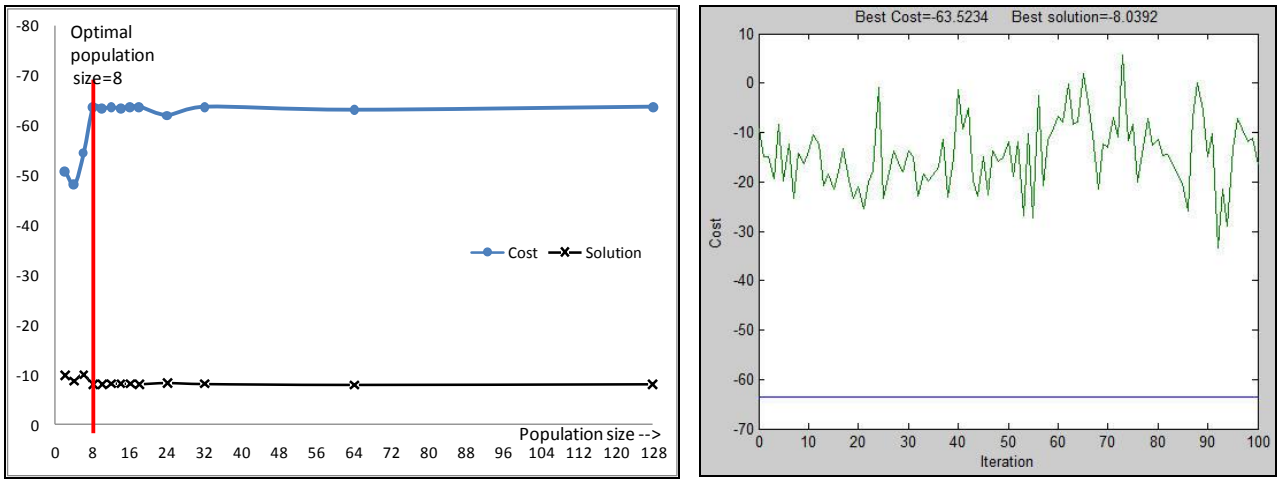
*Fig. 4.***Test function f(x)=x^2*sin(x)**

The goal of the experiment was to find the minimum of the function with the help of GA. The influence of three parameters on the quality of optimization has been investigated:

1. population size;
2. mutation rate;
3. number of bits in parameters.

In the first experiment the initial parameters have been as follows: mutation rate=0.15 and number of bits=8. In all occasions the number of iterations was 100. Results are shown in Figure 5.

The population size has been changed within the boundaries from 2 up to 128. Image a) in Figure 5 shows that the minimum of the function has been reached in cases when the population size is 8, 10, 12, 14, 16. The vertical line shows the optimal population size=8, the value of which has been used further on. It has been established, that the further increase of population size does not especially affect the quality of optimization. Image b) in Figure 5 shows the cost changes in the process of iterations at optimal population size=8.

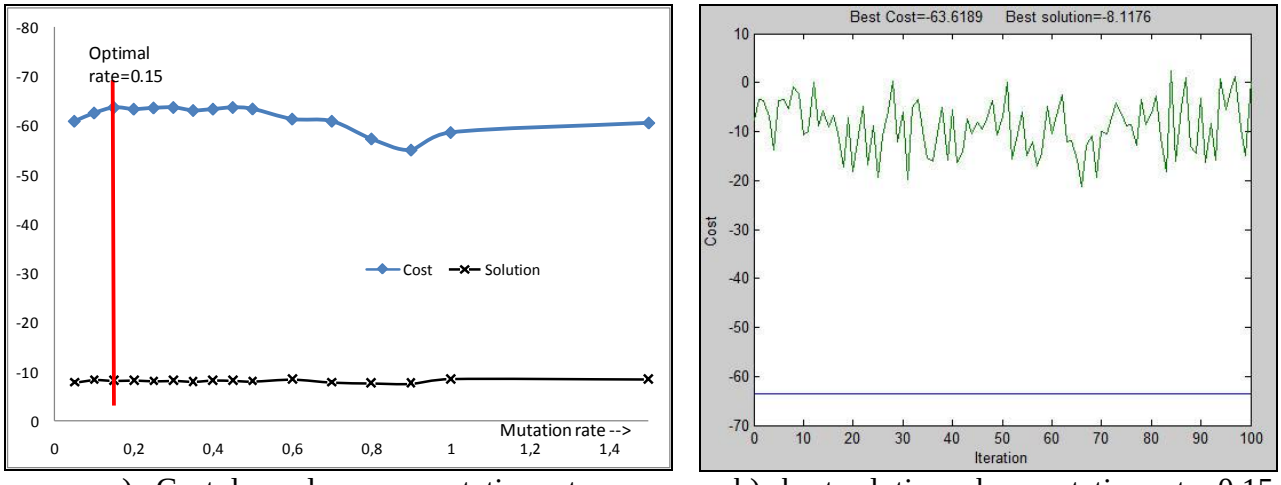


a) Cost dependence on population size b) best solution when population size=8

Fig. 5. **Optimization dependence on parameter *population size***

In the second experiment the initial parameters have been the following: population size=8 and number of bits=8. Results are shown in Figure 6.

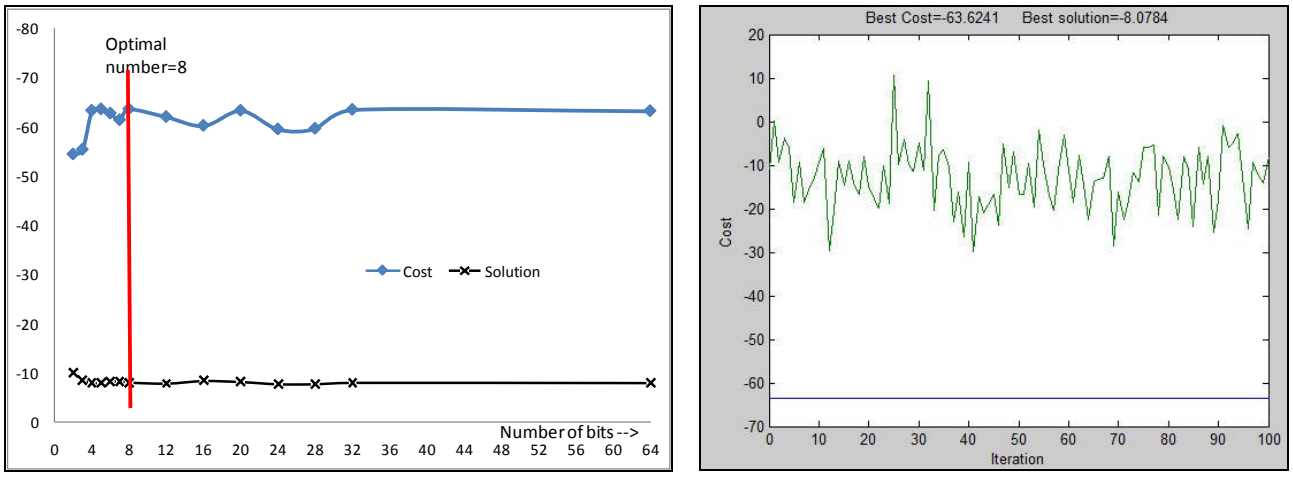
The mutation rate has been changed within the boundaries from 0.05 till 1.5. Image a) in Figure 6 shows that the minimum of the function has been reached in cases, when the mutation rate is from 0.15 till 0.5. The vertical line shows the optimal mutation rate=0.15. It has been established that the further increase of mutation rate does not make it possible to reach the optimal result. Image b) in Figure 6 shows the cost changes in the process of iterations at the mutation rate=0.15.



a) Cost dependence on mutation rate b) best solution when mutation rate=0.15

Fig. 6. **Optimization dependence on parameter *mutation rate***

In the third experiment the initial parameters have been as follows: population size=8 and mutation rate=0.15. Results are shown in Figure 7.



a) Cost dependence on the number of bits b) best solution when number of bits=8

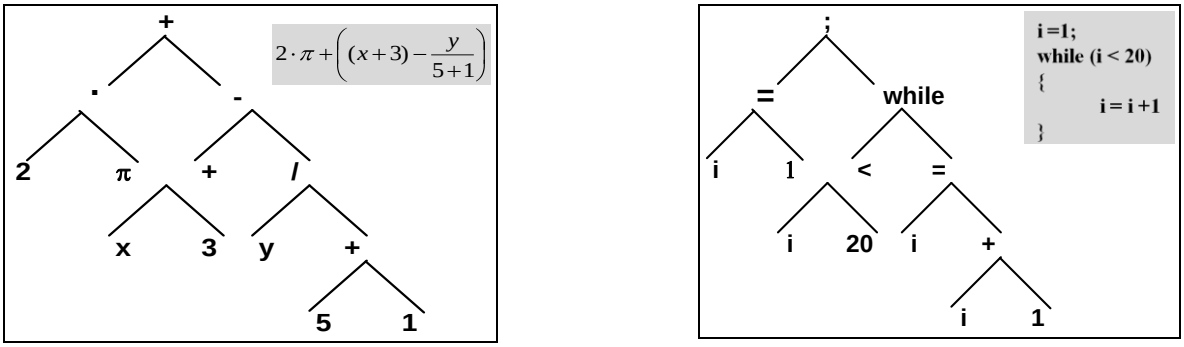
Fig. 7. **Optimization dependence on parameter *number of bits***

The number of bits has been changed within the boundaries from 2 till 64. It can be seen from image a) (Figure 7) that the minimum of the function has been reached at the number of bits=8. Acceptable results have been reached also when the number of bits is 20, 32 and 64. The vertical line shows the optimal number of bits in parameters=8. Image b) in Figure 7 shows the cost changes in the process of iterations at the optimal number of bits=8.
The experiment demonstrates GA usefulness in solving different optimization exercises.

The significance of genetic programming

The term *Genetic Programming* has two possible meanings. First, it is used to subsume all evolutionary algorithms that have tree data structures as genotypes. Second, it can also be defined as the set of all evolutionary algorithms that breed programs, algorithms and similar constructs [8].
The GP approach can be described as $f=f_i+e_i$, where each individual f_i is a function composed recursively from the set $F=\{f_1,...f_{Nf}\}$ of N_f elementary functions and from the set $T=\{t_1,...t_{Nf}\}$ of N_t terminals (e_i - random error term). Terminals can be arithmetic (+, *, /), mathematical (sin, cos), Boolean (and, or, not), conditional (if-then-else), looping (for, repeat). Terminals are operations that take no arguments but return a value (variables or constant values). Beside the variable x, terminals can be random or user defined constants. The fitness of each individual should be a continuous function of the corresponding scalar error.
The main difference between GA and GP is the representation of the solution. GA creates a string of numbers that represent the solution. GP creates computer programs in the scheme computer languages as the solution and individuals are represented as trees.
GP consists of the following four steps:
1) Generate an initial population of random compositions of the functions and terminals of the problem (computer programs).
2) Execute each program in the population and assign it a fitness value according to how well it solves the problem.
3) Create a new population of computer programs (copy the best existing programs, create new computer programs by mutation, create new computer programs by crossover).
4) The best computer program that appeared in any generation, the best-so-far solution, is designated as the result of genetic programming [8].
GP also uses reproduction, crossover, mutation and differs from other EA in the implementation of the operators of crossover and mutation.

In the case of GP search space (phenotypes) is the set of formulas and trees are the natural representation of formulas (genotypes). An example of tree-based representation is shown in Figure 8.



a) arithmetic formula b) program

Fig. 8. **An example of tree-based representation**

A typical application of GP is a symbolic regression. Symbolic regression is the procedure of inducing a symbolic equation, function, or program that fits given numerical data. A GP system performing symbolic regression takes a number of numerical input/output relations, called fitness cases, and produces a function or program that is consistent with these fitness cases [9]. As an illustration the following example has been given.

GP symbolic regression method allows reconstructing of the mathematical function based on the given set of points. In this context, regression is a process in the course of which the reconstruction of the function according to a definite data set takes place. In the method of symbolic regression nor the initial coefficients of the function are known, nor the expression under search is known. One accepts that there are generated expression trees and calculates fitness (see Fig. 9).

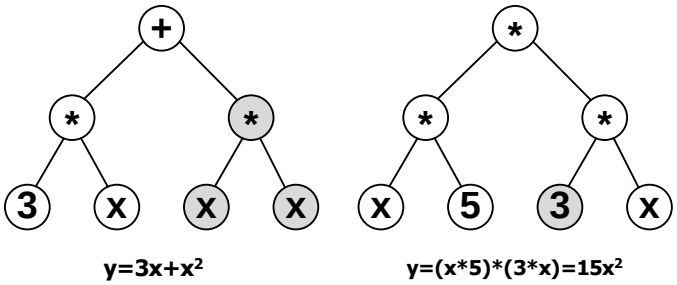


Fig. 9. **Acquired pair of expressions**

Afterwards, from the acquired expressions one chooses sub-trees randomly and thus new trees are obtained (see Fig. 10).

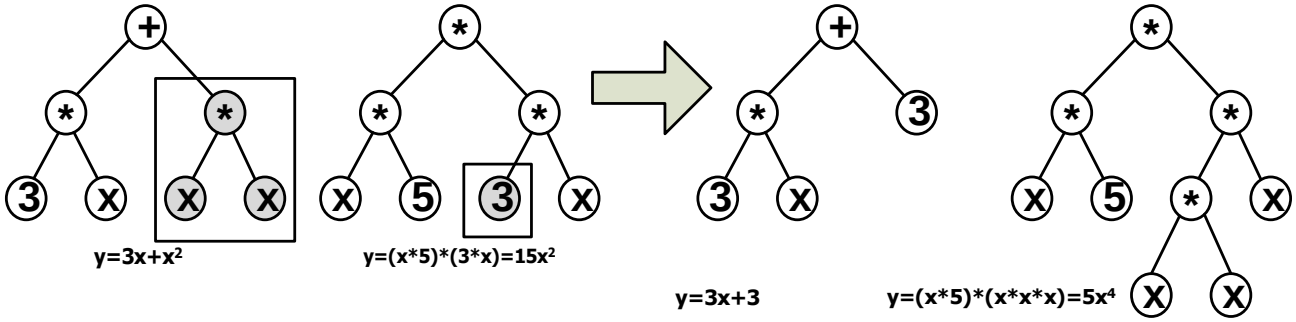


Fig. 10. **Acquired new trees**

On accepting that the fitness of a new tree is the best for the expression y=3x+3, random sub-tree is chosen to mutate (see Fig. 11).

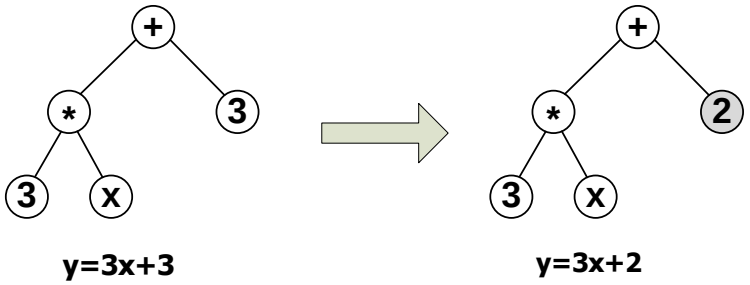


Fig. 11. **Actual data generated using formula y=3x+2+ε**

Thus, the solution has been obtained with the best fitness. Other child tree has a worse fitness. Such methodology is widely applied in constructing of GP trees and is used to solve practical tasks, which will be further investigated in future work.

Conclusions and future work

EA are viewed as a global optimization method although convergence to a global optimum is only guaranteed in a weak probabilistic sense:

- GP are well suited for problems that require the determination of a function that can be simply expressed in a functional form;
- ES and EP are well suited for optimizing continuous functions;
- GA are well suited for optimizing combinatorial problems.

The main advantage of GP is that it performs a global search for a model, contrary to the local greedy search of most traditional machine learning algorithms. Methodology investigated in the paper will be used in further scientific research that will deal with the solutions by using genetic programming.

References

1. Weise T. Global Optimization Algorithms - Theory and Application, 2008. URL: http://www.it-weise.de/index.html - Visit date January 2009.
2. Mitchell M. An introduction to Genetic Algorithms. A Bradford Book The MIT Press, 1999.
3. Introduction on Evolutionary Algorithms. URL: http://neo.lcc.uma.es/opticomm/introea.html - Visit date January 2009.
4. Holland J.H. Adaptation in Natural and Artificial Systems. Univ. of Michigan Press: Ann Arbor. Reprinted in 1992 by MIT Press, Cambridge MA.
5. A Field Guide to Genetic Programming. URL: http://www.gp-field-guide.org.uk/ - Visit date January 2009.
6. Haupt R.L., Haupt S.E. Practical Genetic Algorithms. John Wiley & Sons, 2004.
7. Karr C., Freeman L.M. Industrial Applications of Genetic Algorithms. International Series on Computational Intelligence: CRC Press, 1999.
8. Koza J.R. Genetic Programming: On the Programming of Computers by Means of Natural Selection. Cambridge: MIT Press, 1992.
9. Banzhaf W., Nordin P., Keller R.E. and Francone F.D. Genetic Programming- An Introduction. Morgan Kaufmann Publishers, 1998.