SOCIETY. INTEGRATION. EDUCATION
Proceedings of the International Scientific Conference. Volume V, May 28th-29th, 2021. 330-339

EVOLUTIONARY ALGORITHMS LEARNING METHODS IN STUDENT EDUCATION

Peter Grabusts
Rezekne Academy of Technologies, Latvia

Aleksejs Zorins
Rezekne Academy of Technologies, Latvia

Abstract. *Teaching experience shows that during educational process student perceive graphical information better than analytical relationships. As a possible solution, there could be the use of package Matlab in realization of different algorithms for IT studies. Students are very interested in modern data mining methods, such as artificial neural networks, fuzzy logic, clustering and evolution methods. Series of research were carried out in order to demonstrate the suitability of the Matlab for the purpose of visualization of various simulation models of some data mining disciplines – particularly genetic algorithms. Nowadays the possibilities of evolutionary algorithms are widely used in many optimization and classification tasks. There are four paradigms in the world of evolutionary algorithms: evolutionary programming, evolution strategies, genetic algorithms and genetic programming. This paper analyses present-day approaches of genetic algorithms and genetic programming and examines the possibilities of genetic programming that will be used in further research. Genetic algorithm learning methods are often undeservedly forgotten, although the implementation of their algorithms is relatively strong and can be implemented even for students. In the research part of the study the modelling capabilities in data mining studies were demonstrated based on genetic algorithms and real examples. We assume that students already have prior knowledge of genetic algorithms.*
Keywords: *data analysis, evolutionary algorithms, genetic algorithms, modelling, teaching.*

Introduction

Evolutionary algorithms (EA) are population-based metaheuristic 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 (Koza, 1992, Weise, 2009).

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 judge the quality of these solutions with an objective function and base the selection of surviving solutions on this quality measure.

https://doi.org/10.17770/sie2021vol5.6153

3. They have a reproduction stage in which new solutions are constructed that inherit 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, rather than on the individual. Reproduction is the process how 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. The fittest individuals are those best adapted to their environment, which 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 multiparameter 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) (Mitchell, 1999).

The basic cycle of EA is shown in Figure 1 (Introduction, 2011).

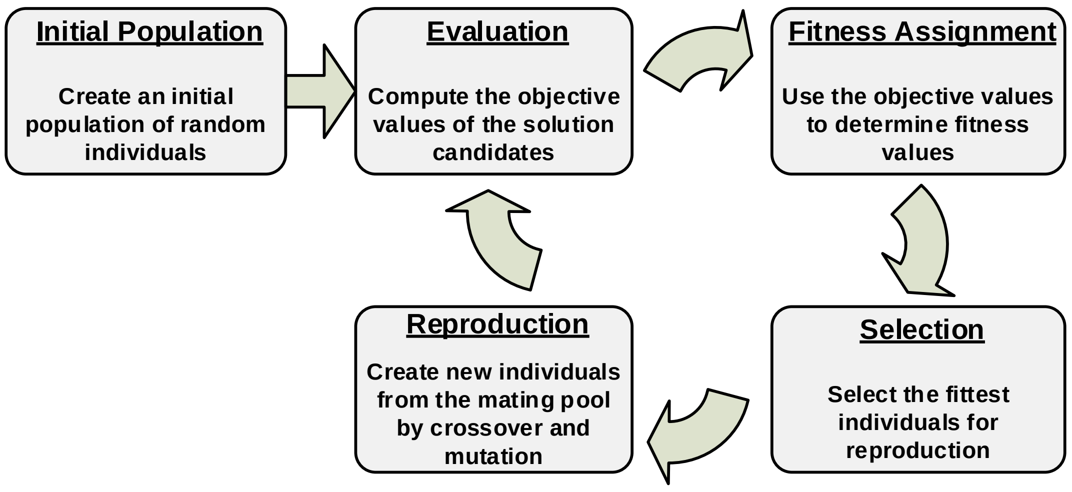
Initial Population
Create an initial population of random individuals
Evaluation
Compute the objective values of the solution candidates
Fitness Assignment
Use the objective values to determine fitness values
Selection
Select the fittest individuals for reproduction
Reproduction
Create new individuals from the mating pool by crossover and mutation

Figure1 ***The Basic Cycle of EA*** *(Introduction, 2011)*

There are several different types of EA (Weise, 2009). These include:

- evolutionary programming (EP), which focuses on optimizing continuous functions without recombination;
- evolutionary strategies (ES), which focuses on optimizing continuous functions with recombination;

- genetic algorithms (GAs), which focuses on optimizing general combinatorial problems;
- genetic programming (GP), which evolve programs.

The Significance of Genetic Programming

Genetic programming is an automated methodology inspired by biological evolution to find computer programs that best perform a user-defined task. It is therefore a particular machine learning technique that uses an evolutionary algorithm to optimize a population of computer programs according to a fitness function determined by a program's ability to perform a given computational task (Goldberg, 1988, Koza, 1992, Mitchell, 1999).

Genetic programming is a methodology for automatically generating computer programs. Rather than writing programs explicitly, GP applies natural selection and genetic recombination to evolve programs that solve a given problem. GP is founded on the premise that computer programs can be represented as tree structures, as shown in Figure 2 (Haupt, R. & Haupt, S., 2004).

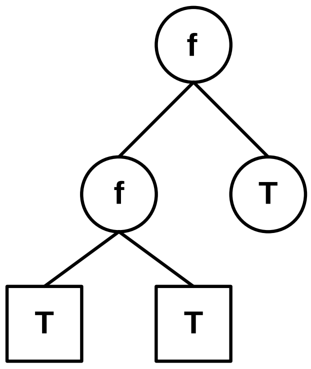

Figure 2 ***Tree Structure Example*** *(Haupt, R. & Haupt, S., 2004)*

The functions (f) operate on terminals (T) to produce a result. Functions are operations that take one or more arguments. They 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).

The main difference between GP and GA is the representation of the solution. GP creates computer programs in the scheme computer languages as the solution. A GA creates a string of numbers that represent the solution. 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;
4) The best computer program that appeared in any generation, the best-so-far solution, is designated as the result of genetic programming (Weise, 2009).

GP difference significantly from other evolutionary algorithms in the implementation of the operators of crossover and mutation.

Mutation is performed by randomly selecting a node in an individual tree structure and removing that node along with any sub-tree that may exist below it. A new sub-tree is then generated randomly and "grafted in" at the position where the original node was removed. Example of sub-tree mutation is illustrated in Figure 3 (Introduction, 2011; Buontempo, 2019).

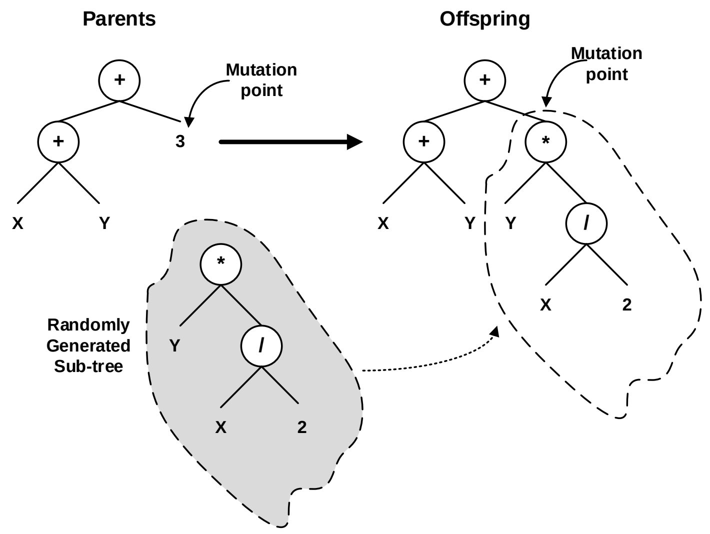

Figure 3 ***An Example of Sub-tree Mutation*** *(Buontempo, 2019)*

Crossover involves selecting two individuals from the previous generation and selecting a node at random in each of them. The selected nodes, along with any sub-trees that exist below them, are exchanged between the two individuals.

There is no guarantee that GP will find an optimal solution, but a well thought out set of functions with a reasonable fitness test will usually produce good results (Mitchell, 1999, Karr & Freeman, 1999).

Decision Tree Representation for GP

Decision trees and decision rules are data mining methodologies applied in many applications as a powerful solution to classification problems. In general, classification is a process of learning a function that maps a data item into one of several predefined classes.

A decision tree representation would be able to correctly handle both numerical and categorical values. Numerical variables and values should only be compared to numerical values or variables and only be used in numerical functions. Similarly, categorical variables and values should only be compared to categorical variables or values. This is a problem for the standard GP operators (crossover, mutation and initialization) which assume that the output of any node can be used as the input of any other node. This is called the closure property of GP which ensures that only syntactically valid trees are created.

A solution to the closure property problem of GP is to use strongly typed genetic programming. Strongly typed GP uses special initialization, mutation and crossover operators. These special operators make sure that each generated tree is syntactically correct even if tree-nodes of different data types are used. Because of these special operators an extensive function set consisting of arithmetic (+, −, ×, /), comparison (≤, >) and logical operators (and, or, if) can be used.

Another strongly typed GP representation was introduced in (Bot & Langdon, 2000). This linear classification GP algorithm uses a representation for oblique decision trees. An example tree can be seen in Figure 4.

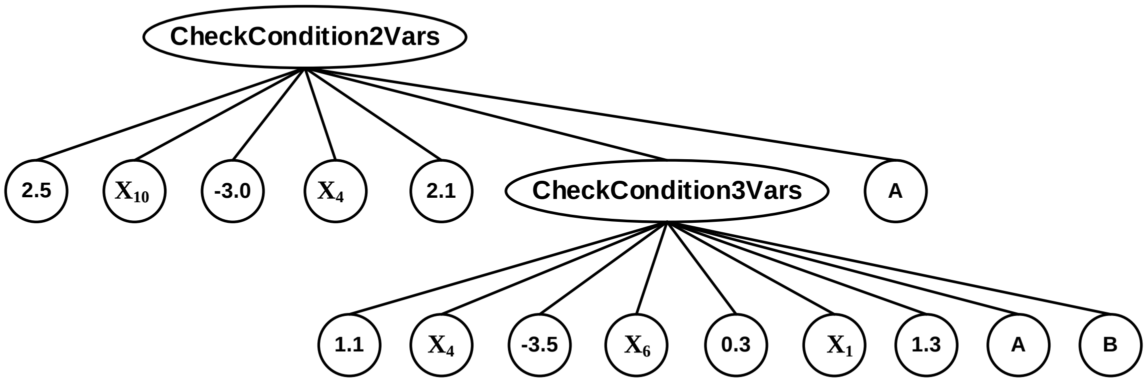

Figure 4 ***Example Decision Tree and Its Representation in the GP*** *(Bot & Langdon, 2000)*

The leftmost children of function nodes (in this case *CheckCondition2Vars* and *CheckCondition3Vars*) are weights and variables for a linear combination. The rightmost children are other function nodes or target classes (in this case A or B). Function node *CheckCondition2Vars* is evaluated as: if 2.5x10 − 3.0x4 ≤ 2.1 then evaluate the *CheckCondition3Vars* node in a similar way; otherwise the final classification is A and the evaluation of the decision tree on this particular case is finished.

In 1998 a new representation was introduced (Hemert, 1998) - atomic representation booleanizes all attribute values in the terminal set using atoms. Each atom is syntactically a predicate of the form (variable operator constant) where operator is a comparison operator (e.g., ≤ and > for continuous attributes, = for nominal or Boolean attributes). Since the leaf nodes always return a Boolean value (true or false) the function set consists of Boolean functions (e.g., and, or) and possibly a decision-making function (if − then − else) (Hemert, 1998). An example of a decision tree using the atomic representation can be seen in Figure 5. Input variables are booleanized using atoms in the leaf nodes. The internal nodes consist of Boolean functions and possibly a decision-making function.

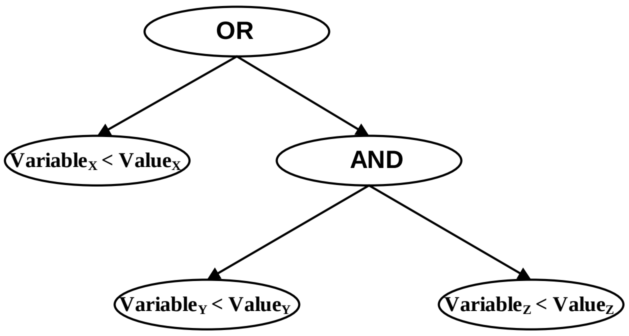

Figure 5 ***An Example of a Decision Tree Using an Atomic Representation*** *(Hemert, 1998)*

In conclusion there is many different possibilities for the representation of decision trees.

Experimental Part

A function was selected to illustrate GA operation f(x)=x2*sin(x) with minimum: f (-8.0962) =-63.635 for -10≤x≤10 (see Fig. 6). Best cost=-63.635 and best solution=-8.0962.

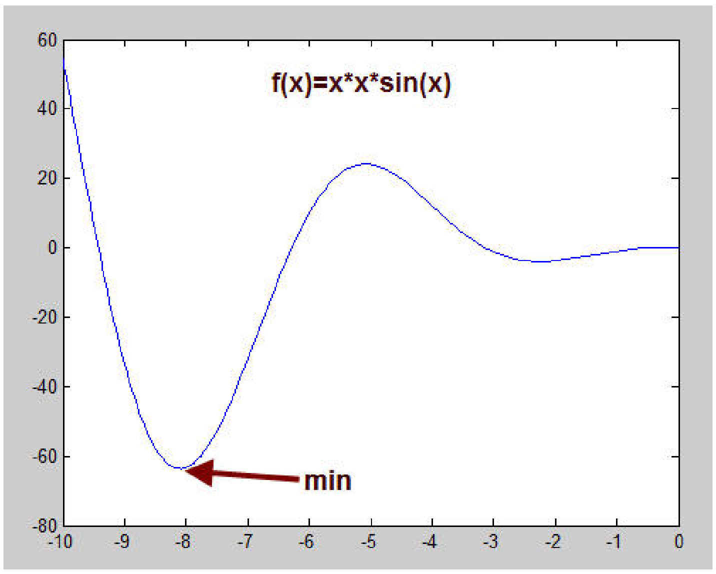

Figure 6 ***Test Function f(x)=x²*sin(x)***

The aim of the experiment was to find the minimum of the function with the help of GA (Grabusts, 2009).The effect of three parameters on the quality of optimization was studied:

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

In the first experiment, the initial parameters were as follows: mutation rate=0.15 and number of bits=8. In all cases, the number of iterations was 100.

Population size was varied from 2 to 128. Figure 7 shows that the minimum of the function is stably 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 was used below. It was found that further increase in population size does not significantly affect the quality of optimization. Figure 8 shows the cost changes in the iteration process at optimal population size = 8.

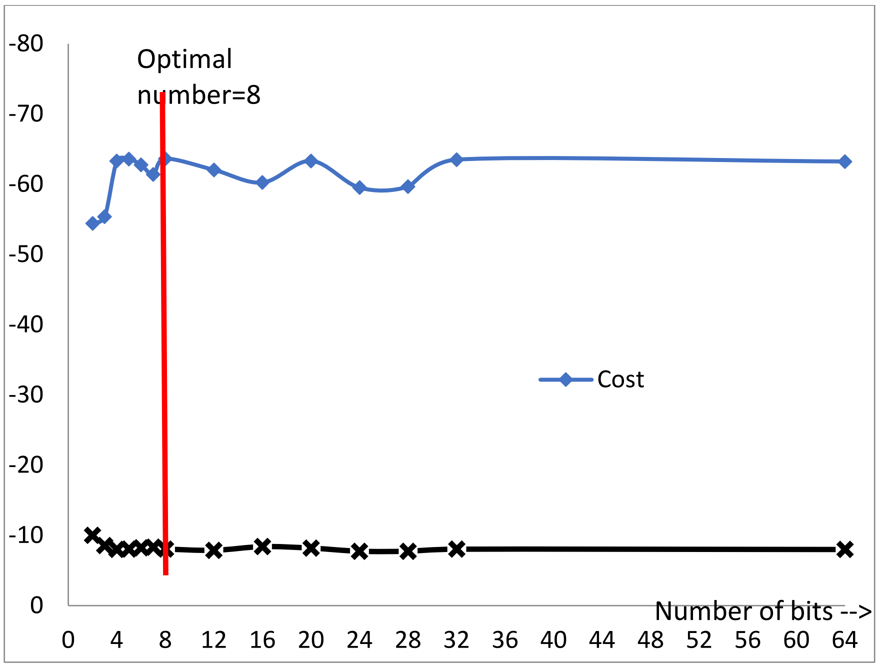

Figure 7 ***Dependence of Cost on Population Size***

The mutation rate was ranged from 0.05 to 1.5. The minimum of the function is stably reached in cases when the mutation rate is from 0.15 to 0.5. The vertical line shows the optimal mutation rate = 0.15 (Fig. 9). It was found that further increase of the mutation rate does not give the optimal result.

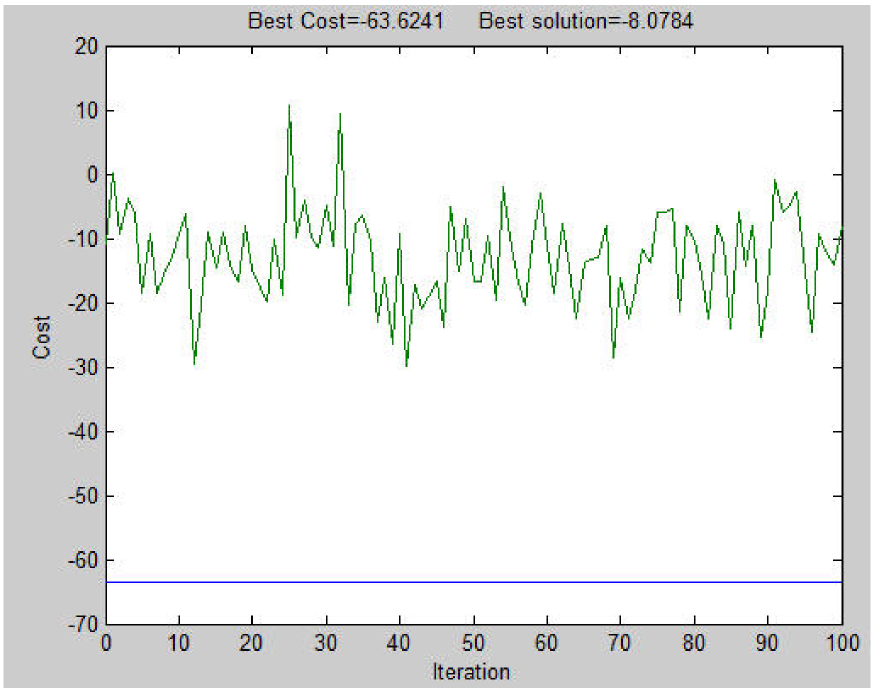

Figure 8 ***Best Solution When Population Size=8***

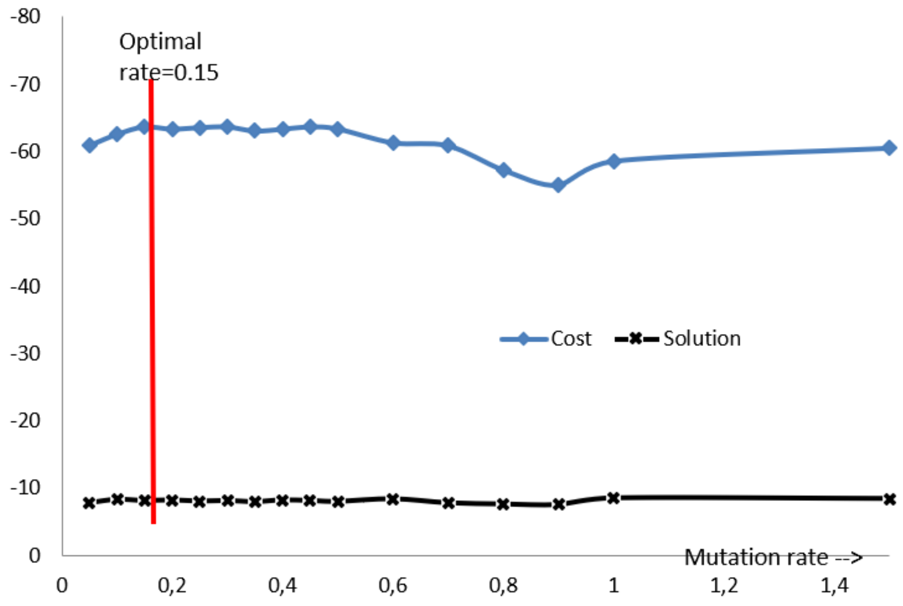

Figure 9 ***Dependence of Cost on Mutation Rate =0.15***

The experiments clearly demonstrate the usefulness of GA in solving various optimization tasks for educational research (Grabusts, 2009).

Conclusions

The term GP 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 (Gupta & Sinha, 2020).

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.

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. To design the decision tree using GP, everyone is defined as a decision tree, which represents both the genotype and the phenotype.

The methodology investigated in the work will be used in further scientific research for student's investigations that will deal with design the decision tree using genetic programming.

This paper analyses present-day approaches of GA and GP and examines the possibilities of GP that will be used in further student's research. GA learning methods are often undeservedly forgotten, although the implementation of their algorithms is relatively strong and can be implemented even for students in this future scientific research.

References

Bot, M.C., & Langdon, W.B. (2000). Application of Genetic Programming to Induction of Linear Classification Trees. *Proceedings of the Third European Conference on Genetic Programming.* Received from: https://link.springer.com/chapter/10.1007/978-3-540-46239-2_18

Buontempo, F. (2019). *Genetic Algorithms and Machine Learning for Programmers: Create AI Models and Evolve Solutions*. The Pragmatic Programmers, 225p.

Goldberg, D. (1988). *Genetic Algorithms in Search, Optimization and Machine Learning*. 13th ed. Edition. Addison-Wesley Professional, 432p.

Grabusts, P. (2009). Evolutionary algorithms at choice: From GA to GP. *7th International Scientific and Practical Conference "Environment, Technology and Resources", Rezekne. Volume 2, 2009*, 185-192.

Gupta, S., & Sinha, S. (2020). Academic Staff planning, allocation and optimization using Genetic Algorithm under the framework of Fuzzy Goal Programming. *Procedia Computer Science*, 172. Retrieved from: https://www.sciencedirect.com/science/article/pii/S1877050920314599

Haupt, R.L., & Haupt S.E. (2004). *Practical Genetic Algorithms*. John Wiley & Sons.

Introduction on Evolutionary Algorithms. (2011). Retrieved from: https://neo.lcc.uma.es/opticomm/introea.html

Karr, C., & Freeman, L.M. (1999). *Industrial Applications of Genetic Algorithms*. International Series on Computational Intelligence: CRC Press.

Koza, J.R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge: MIT Press.

Mitchell, M. (1999). *An introduction to Genetic Algorithms*. A Bradford Book. The MIT Press.

Hemert, J.I. (1998). *Applying Adaptive Evolution Algorithms to Hard Problems*. Master Thesis. Leiden University.

Weise, T. (2009). *Global Optimization Algorithms - Theory and Application*. Retrieved from: http://www.it-weise.de/projects/book.pdf