Chapter Number

Two-Level Transplant Evolution for Optimization of General Controllers

Roman Weisser and Pavel Ošmera
University of Technology VUT in Brno
European Polytechnical Institute Kunovice
Czech Republic

1. Introduction

The aim of this paper is to describe a new optimization method that can create control equations of general regulators. For this type of optimization a new method was created and we call it Two-Level Transplant Evolution (TLTE). This method allowed us to apply advanced methods of optimization, for example direct tree reducing of tree structure of control equation. The reduction method was named Arithmetic Tree Reducing (ART). For optimization of control equations of general controllers is suitable combine two evolutionary algorithms. Main goal in the first level of TLTE is the optimization of structure of general controllers. In the second level of TLTE the concrete parameters are optimized and the unknown abstract parameters in structure of equations are set. The method TLTE was created by combination of Transplant Evolution method (TE) and Differential Evolution method (DE). The Transplant Evolution (TE) optimizes structure of solution with unknown abstract parameters and the DE optimizes the parameters in this structure. The parameters are real numbers. The real numbers are not easy find directly in TE without DE. For evaluation of quality of found control equation are described new methods, which allow us evaluate their quality. It can be used in the case when the simulation of control process cannot be finished. In results are shown some practical application. In all results we received the control equation that reached better quality of control process, than classical PSD controllers and Takahashi`s modification of PSD controller.

2. Transplant Evolution

Transplant Evolution (TE) was inspired by biological transplantation. It is analogy as transplant surgery of organs. Every transplanted organ was created by DNA information but some parts of ill body can be replaced by a new organ from database of organs. The description parts of individual (organs) in the database are not stored on the level DNA (genotype). In the Transplant Evolution (TE) every individual part (organ) is created by translation of grammar rules similar as in Grammatical Evolution (GE), but Transplant Evolution (TE) does not store the genotype and grammatical rules are chosen randomly. The new created structure is stored only as tree. This is like Genetic Programming (GP).
The Transplant Evolution algorithm (TE) combines the best properties of Genetic Programming (GP) [2] and Grammatical Evolution (GE) [4]. The Two-Level Transplant

Evolution (TLTE) in addition to that uses Differential Evolution algorithm (DE). Optimization of numerical parameters of general controllers in recurrent control equations of general controllers is very difficult problem during which is applied the second level of optimization by Differential Evolution method (DE). The individuals in TE and TLTE are represented only by phenotype in shape of object tree. During the initialization of population and during of creation these phenotypes, the similar methods are used as in GE. In Grammatical Evolution are used the numerical represented chromosomes. The codons of these chromosomes are used for selection some rule from set of rules. In the Transplant Evolution is not use the chromosome and codons, but for selection some rule from set of rules are use random generated numbers. These numbers are not stored in individual chromosome. The new individuals in population are created with use analytic and generative grammar and with uses crossover and mutation operators. These operators are projected for work with phenotype of individual, similarly as in GP. Because individuals of TE and TLTE are represented only phenotype, it was possible to implement advanced methods in the course of evolution:

- effective change of rules in the course of evolution, without loss of generated solutions,
- differnent of probability of selection rules from set of rules and possibility of its changing during the evolutionary process,
- the possibility of using methods of direct reduction of the tree using algebraic operations,
- there is a possible to insert some solutions into the population, in the form of an inline entry of phenotype (for example: $U_{k-1} + E_k + E_{k-1} * num$),
- a new methods of crossover is possible to use, (for example *crossover by linking trees*)
- etc.

2.1 Initialization of individual

During the initialization, the generative grammar rules are used. These rules are selected randomly from set of rules by the following equation:

$$rule = random(0, \max \mathrm{Int}) \% rules\_count \tag{1}$$

Where: *random* is pseudo-random number generator, *maxInt* is a high number, % is remainder operator (modulus), *rules_count* denotes the number of possible rules to transcribe given non-terminal symbol.

The algorithm TE which is described by equation 1 differs by the way of initialization of individual in Grammatical Evolution (GE). Original initialization algorithm GE uses forward processing of grammatical rules. In the Grammatical Evolution the method of crossover and mutation are made on genotype level. The phenotype is created by later translation of this genotype. This way of mutation and crossover does not allow using of advanced method in crossover and mutation operators, which does not destruct already created parts of the individual (O'Neill M. and Ryan C., 2003). During evolution of this algorithm the Backward Processing of Rules (BPR) arose (Popelka, O., 2007). The BPR method use gene marking. This approach has hidden knowledge of tree structure. Due to BPR method can use advanced and effective methods of crossover and mutation. Anyhow BPR method has some disadvantages. Due to this disadvantages the new method Transplant Evolution (TE) was created. The TE method does not store genotype information. The equation 1 is used for selection of rule from rules base. The advantage of

TE is possibility use of both types of grammatical processing (forward and backward) with same results, because TE works only with phenotype and procedure of phenotype creation is not important. Some examples of forward and backward initializations are shown in Fig. 1 and Fig. 2. The phenotype in these cases has the following structure $U_{k-1} + 2* E_k + 5 * E_{k-1}$.

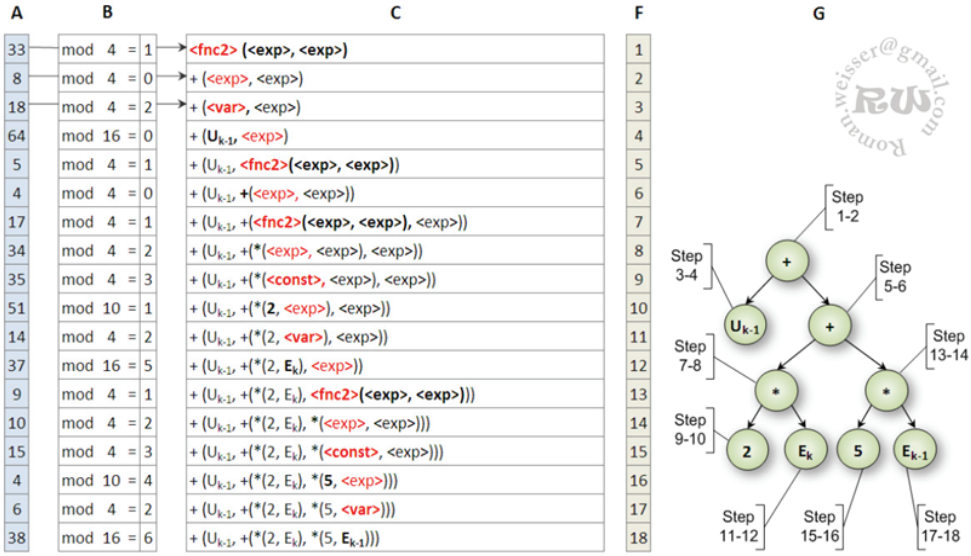

Fig. 1. Translation in TE (forward processing)

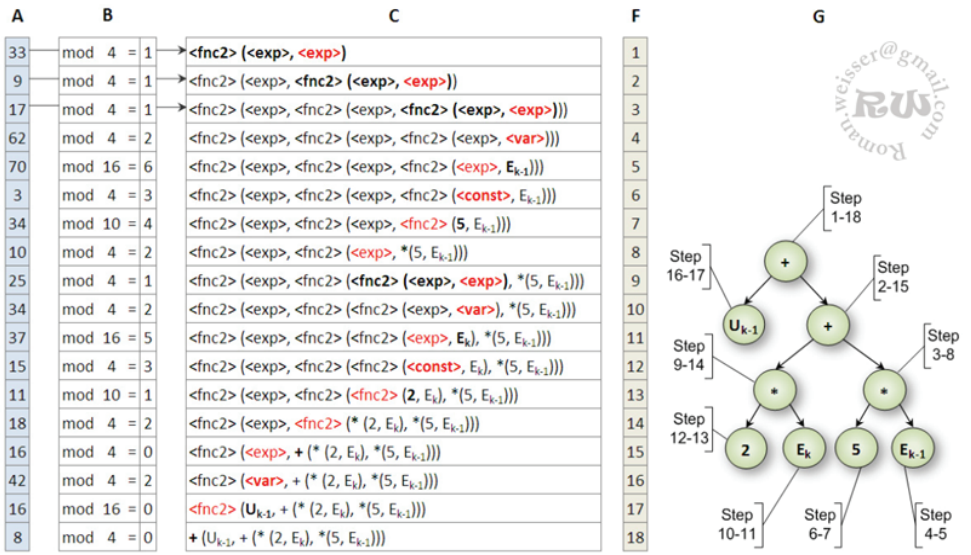

Fig. 2. Translation in TE (backward processing)

In the *column A* are shown randomly generated values. These values are not stored anywhere! Values are generated only when necessary select from more grammatical rules. In the *column B* is shown arithmetic operation. In the *column C* is shown a state of rules translation, but it must be remembered that the translation rules is done at the tree (in the nodes of phenotype). In the *column F* is the order of operations. These numbers are used for description of tree nodes initialization order. The tree initialization is shown at *column G*. Each node of tree in the *column G* is described by number in form *Step X-Y*. In the ***Step X-Y***, *X* represents a step in which the node was created. *Y* represents a step in which the node was fully initialized (after initialization all of its subnodes).
As you can see, the finite tree structures are the same, but they have different order of fully initialization. The generated numbers in column A was changed too.
The both examples of initialization of individual expected generative grammar rules (productive rules) in prefix form. The generative grammar is defined in Table 1. The rules in this table are written in Backus Naur Form (BNF) (BACKUS, J. W., et al., 1997).

$\langle S \rangle$	::=	$\langle exp \rangle$
$\langle exp \rangle$	::=	$\langle fnc1 \rangle \langle exp \rangle \mid \langle fnc2 \rangle \langle exp \rangle \langle exp \rangle \mid \langle var \rangle \mid \langle const \rangle$
$\langle fnc1 \rangle$	::=	$U+ \mid U-$
$\langle fnc2 \rangle$	::=	$+ \mid - \mid * \mid /$
$\langle var \rangle$	::=	$U_{k-1} \mid U_{k-2} \mid U_{k-3} \mid U_{k-4} \mid U_{k-5} \mid$ $E_k \mid E_{k-1} \mid E_{k-2} \mid E_{k-3} \mid E_{k-4} \mid E_{k-5} \mid$ $dE_k \mid dE_{k-1} \mid dE_{k-2} \mid dE_{k-3} \mid dE_{k-4}$
$\langle const \rangle$	::=	$1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \mid 0$

Table 1. Generative grammar

The main principle of initialization of individual in Transplant Evolution is described by recurrent pseudo-code in Fig. 3. In this pseudo-code can be seen that some rules can be disabled or the probability of selection can be changed. This possibility can be seen on row 2 in Fig. 3. The random selection of rule from rule base are shown on row 9. This approach is realized by method called Get_random_rule. The method CreateSubTree on row 11 create the subtree of present tree.

2.2 Crossover

The crossover is a distinctive tool for genetic algorithms and is one of the methods in evolutionary algorithms that are able to acquire a new population of individuals. The crossover is realized similar way as in Genetic Programming (GP) [2].
The TE uses three types of crossover. The first type of crossover is named *crossover the parts of trees*, the second type is named *crossover of nodes*, and the third type is named *crossover by linking trees*. The nodes or subparts of trees in the individuals for crossover are selected randomly.
In the case of the method *crossover of nodes* is neccessary to keep change of same types of nodes. These request is possibly to express by this relationship:

$$\langle fnc1 \rangle \nLeftrightarrow \langle fnc2 \rangle \nLeftrightarrow \{\langle \text{var} \rangle \Leftrightarrow \langle const \rangle\} \tag{2}$$

The method of *crossover by linking trees* is a new method. This method creates one offspring from two parents. The parts of parents are linked by the newly generated node. The newly

generated node has two subparts. The one of them is tree of first parent and the second of them is tree of second parent.

```
Input: G – context-free grammar; P – array of rules; L_max – max nodes count
Result: individual
1  if length[this] ≥ L_max then
2  |  Disable_rules('function') ;
3  end
4  for i ← length[P] to i ≥ 1 do
5  |  if P[i] is terminal symbol then
6  |  |  rule ← P[i]
7  |  else
8  |  |  P_temp ← array of rules for the nonterminal P[i] in G indexed from zero;
9  |  |  rule ← Get_random_rule(P_temp) ;
10 |  end
11 |  sub_tree ← CreateSubTree(G; rule; L_max) ;
12 |  this ← add child sub_tree ;
13 end
14 if P[0] is terminal then
15 |  this.terminal = P[0] ;
16 else
17 |  sub_tree ← CreateSubTree(G; P[0]; L_max) ;
18 |  this.terminal ← sub_tree;
19 end
20 return this;
```

Fig. 3. Pseudo-code of initialization of individuals

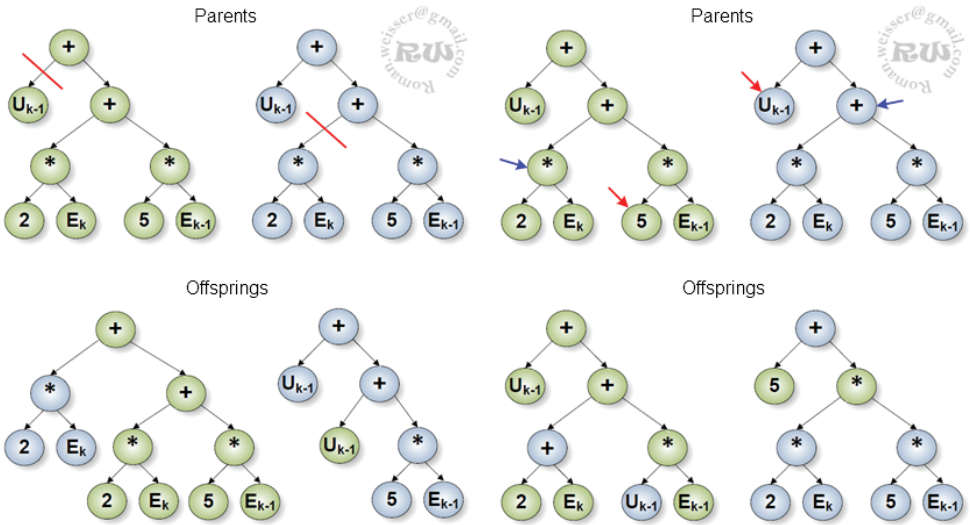

Fig. 4. Crossover: a) crossover the parts of trees, b) crossover of nodes

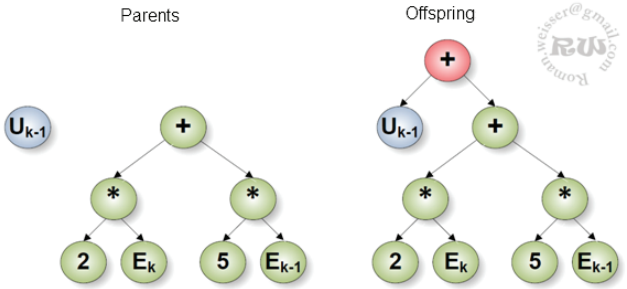

Fig. 5. Crossover by linking trees

2.3 Mutation

Mutation is the second operator to obtain new individuals. This operator can add new structures, which are not included in the population so far. Mutation is performed on individuals from the old population. The nodes in the individuals for mutation are selected randomly. The mutation operator can be subdivided into two types. The first type of mutation is *non-structural mutation* and second type is *structural mutation*. Structural mutation can be divided into shortening structural mutation and extending structural mutation. The mutation operator uses analytic grammar and generative grammar rules.

Non-structural mutation do not affect the structure of already generated individual. In the individual that is selected for mutation, chosen nodes of object sub-tree are further subjected to mutation. The mutation will randomly change chosen nodes, whereas used grammar is respected. For example it means that mutated node, which is a function of two variables (i.e. + - × ÷) cannot be changed by node representing function of one variable (unary minus, etc.) or only a variable (Ek, etc.), etc. See Fig. 6.

Selected node in the tree G1 is marked by orange color (or different level of gray). The tree after mutation is marked G2 and mutated node is marked orange color (or different level of gray) too. The mutation was made by using of generative and analytic grammar. The operations with this grammar are marked in columns A, B, C, and F. The randomly generated value for rule selection are shown in column A. Modulus operation are shown in column B. The processing of rules is shown in column C.

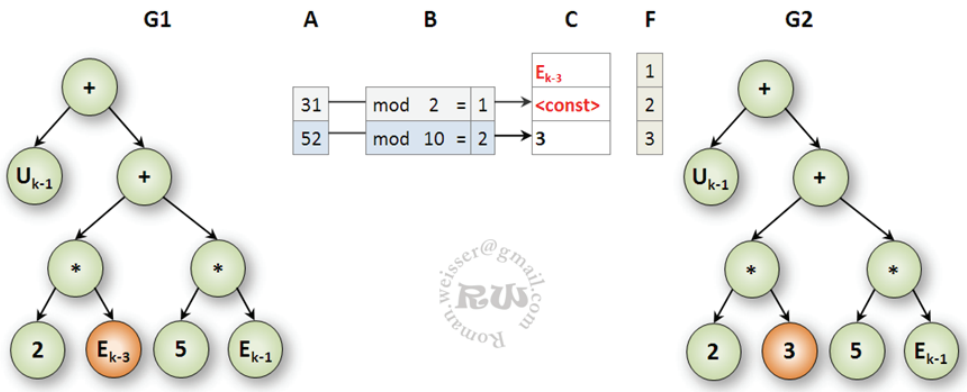

Fig. 6. Non-structural mutation

This example assumes generative grammar rules in Table 1 and analytic grammar rules in Table 2.

$U_{k-1} \mid U_{k-2} \mid U_{k-3} \mid U_{k-4} \mid U_{k-5}$	::=	$\langle var \rangle \mid \langle const \rangle$	
$E_k \mid E_{k-1} \mid E_{k-2} \mid E_{k-3} \mid E_{k-4} \mid E_{k-5}$	::=	$\langle var \rangle \mid \langle const \rangle$	
$dE_k \mid dE_{k-1} \mid dE_{k-2} \mid dE_{k-3} \mid dE_{k-4}$	::=	$\langle var \rangle \mid \langle const \rangle$	
$decimal$	::=	$\langle var \rangle \mid \langle const \rangle$	
$U+ \mid U-$	::=	$\langle fnc1 \rangle$	
$+ \mid - \mid * \mid /$	::=	$\langle fnc2 \rangle$	

Table 2. Analytic grammar (non-structural mutation)

Structural mutation, unlike non-structural mutations, affect the tree structure of individuals. Changes of the sub-tree by extending or shortening its parts depend on the method of structural mutations. Structural mutation can be divided into two types: Structural mutation which is *extending an object tree structure* and structural mutation which is *shortening a tree structure*. In the case of the **extending structural mutation**, a randomly selected node is replaced by a part of the newly created sub-tree that respects the rules of defined grammar (see Fig. 7). Conversely the **shortening structural mutation** replaces a randomly selected node of the tree, including its child nodes, by node which is described by terminal symbol (i.e. a variable or a number). This type of mutation can be regarded as a method of indirectly reducing the complexity of the object tree (see Fig. 7).

On this figure are presented two trees, which are marked G1 and G2. In the case of shortening mutation the mutation is done from tree G1 to tree G2, through operations which are shown in columns A, B, C, and F (above the arrows). In the case of extending mutation the mutation is done from tree G2 to tree G1, through operations which are shown in columns A, B, C, and F (under arrows). Randomly generated values for selection of the rules are shown in column A, arithmetic operation modulus is shown in column B, the processing of rules is shown in column C, the order of operation is shown in column F.

The operator of structural mutation in Fig. 7 assumes the analytic grammar and generative grammar rules from tables Table 3, Table 4, Table 5 and Table 1.

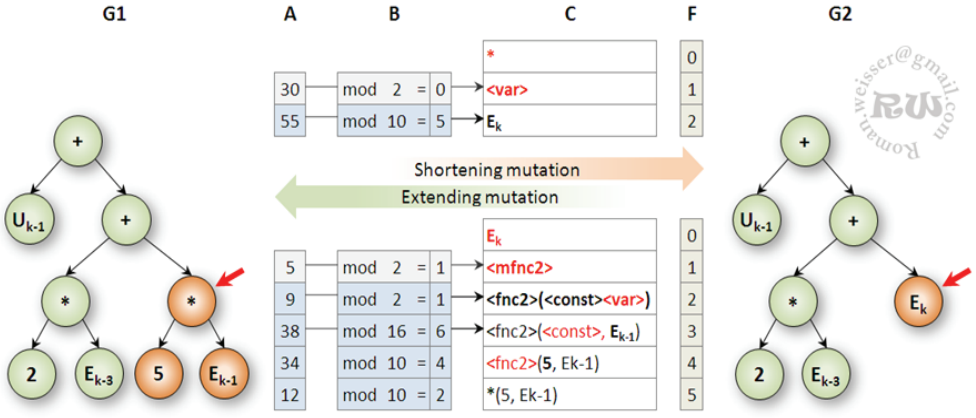

Fig. 7. Structural mutation (Shortening, Extending)

$U_{k-1} \mid U_{k-2} \mid U_{k-3} \mid U_{k-4} \mid U_{k-5}$	::=	$\langle mfnc1 \rangle \mid \langle mfnc2 \rangle$
$E_k \mid E_{k-1} \mid E_{k-2} \mid E_{k-3} \mid E_{k-4} \mid E_{k-5}$	::=	$\langle mfnc1 \rangle \mid \langle mfnc2 \rangle$
$dE_k \mid dE_{k-1} \mid dE_{k-2} \mid dE_{k-3} \mid dE_{k-4}$	::=	$\langle mfnc1 \rangle \mid \langle mfnc2 \rangle$
$U+ \mid U-$	::=	$\langle mfnc2 \rangle$

Table 3. Analytic grammar (extending structural mutation)

$U+ \mid U-$	::=	$\langle var \rangle \mid \langle const \rangle$
$+ \mid - \mid * \mid /$	::=	$\langle var \rangle \mid \langle const \rangle$

Table 4. Analytic grammar (shortening structural mutation)

$\langle mfnc1 \rangle$	::=	$\langle fnc1 \rangle \langle var \rangle \mid \langle fnc1 \rangle \langle const \rangle$
$\langle mfnc2 \rangle$	::=	$\langle fnc2 \rangle \langle var \rangle \langle var \rangle \mid \langle fnc2 \rangle \langle const \rangle \langle var \rangle$

Table 5. Generative grammar (expansion for analytic grammar)

3. Two-level Transplant Evolution

The Two-Level Transplant Evolution (TLTE) is new type of evolutionary algorithm that performs the optimization of structure in the first level and the parameters in the structure are optimized in the second level of optimization. The optimization of structure is realized by the Transplant Evolution (TE) and the optimization of abstract numerical parameters are realized by the Differential Evolution (DE) [6], or some other algorithm for numerical parameters optimization. The join the two evolution algorithms has the goal eliminate very hard optimization of numerical parameters by the grammar based evolutionary algorithms. The DE method was chosen for high power of optimization the real number parameters.
The generative grammar and analytic grammar rules are need to modify for generate or mutation of abstract numerical parameters. To these grammar rules is necessary to add these expansion rules.

$\langle var \rangle$	::=	num

num	::=	$\langle var \rangle$

Table 6. Generative and analytic grammar (expansion for TLTE)

Some early defined rules are possible to delete. For example non-terminal <const>, etc.

3.1 Architecture

From implementation look, the join two evolutionary algorithms is the most difficult process. For best flexibility was chosen interconnecting three separate computational module that are join by interlayer. The interlayer mediates a communication among modules and prepares data in require form. With this implementation is possible to use each module separately for another optimization problem, or similarly change some module by another module.

3.1.1 Transplant Evolution Module

Transplant Evolution Module (MTE), see Fig. 8 ensures optimization of the structure of controllers. The principle of this algorithm was described in article TLTE. Its task is to create

a population of individuals which uses generative grammar G and analytic grammar G^{-1}. This module can make a numerical optimization of parameters in the structure solutions too, but the quality is not so good as in the case of two-level structure. Optimization of parameters is provided by Differential Evolution Module (MDE). The agent for communication between these layers is Interlayer, which is responsible for preparing the data in the desired form for each module. Input parameters for the MTE is generative grammar rules P and analytic grammar rules P^{-1}, which leads to evolution of individuals (solutions). The output parameter of the MTE is a fully optimized solution that includes both the appropriate structure, and its parameters.

3.1.2 Differential Evolution Module

The Differential Evolution Module (MDE), see Fig. 8 ensures the optimization of abstract parameters in the structure of the individual. The principle function of *Differential Evolution* (DE) is explained for example in [4]. This module is activated by the *Interlayer*, which gives him the vector of variables $\overline{num} = (num_1, ..., num_n)$ as one of input parameters. Number of parameters of this vector is equal to the number of variables, which was included in the structure of the solutions that came from the MTE. Output parameter MDE is the optimized vector of parameters that is send into the *Interlayer*.

3.1.3 Fitness Module

Fitness Module (FM), see Fig. 8, provides evaluation of models. The model is some structure of solutions, including some parameters. This model, together with the controlled system is the input of simulation. For a given controller and controlled system, the simulation of regulation is started. The quality of model is included in fithess function after finishing of the simulation. The evaluated model is send to *Interlayer*.

3.1.4 Interlayer

The Interlayer, see Fig. 8 is most important in the whole architecture of Two-Level architecture. The Interlayer is the main module, which holds instances of all computational modules. The *interlayer* provides communication among computational modules through communication interface. This module contains the knowledge about the strategy optimization it means what to be optimized and how to optimized it. The *interlayer* controls the parameters of all computational modules (MTE, MDE, and FM), for example: size of populations TE or DE, number of generations, type of selection methods, set of probability of crossover and mutation, etc. The *interlayer* makes the communication with Output Module (OM). The OM is graphical output, which shows evolution outputs, simulation of control outputs, etc. The most important task of *interlayer* is data preparation in the correct form during the communication among the computational modules, for example during the request for fitness evaluation, see Fig. 8. From this perspective, the tasks can be divided as follows:

- Create the vector of symbolic variables for MDE during the communication with MTE.
- The abstract variables will be set by concrete optimized parameters in abstract variables after finishing of optimization.
- Send full model of solution (structure + parameters) to FM, during the request of MTE for calculation of quality of individual, which does not include any abstract parameter.

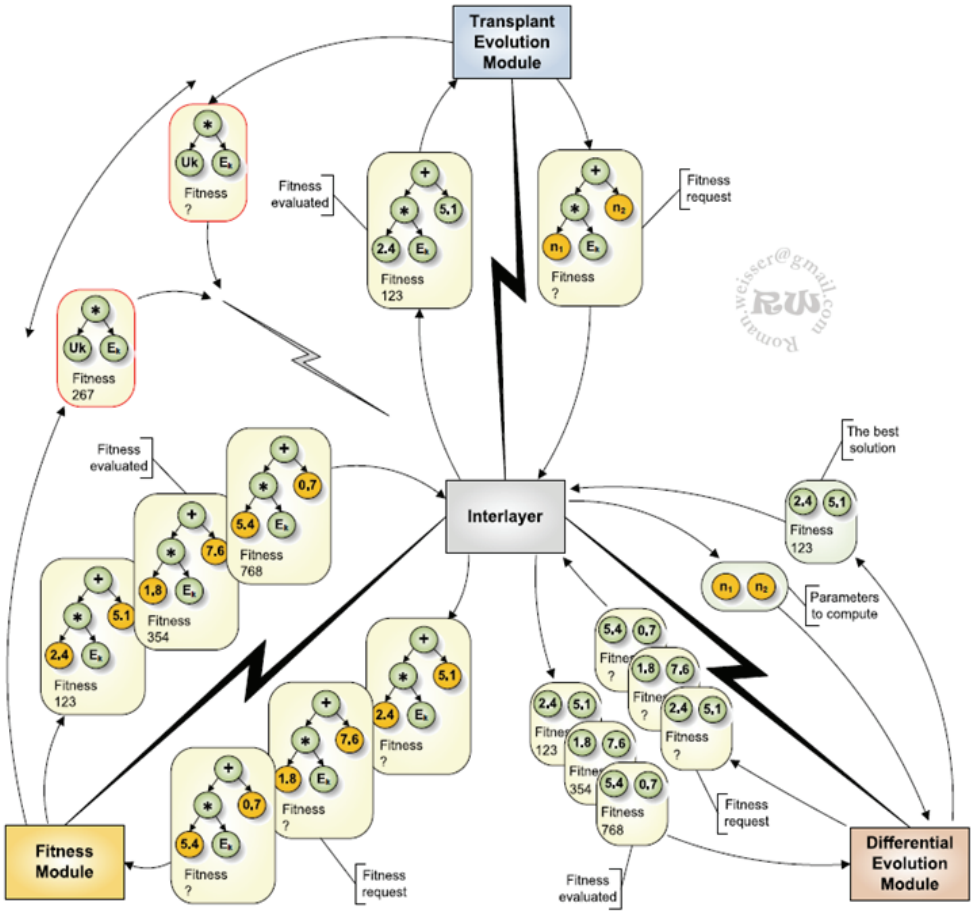

Fig. 8. Architecture and data flow in TLTE

3.2 Arithmetic tree reducing

The minimal length of an object tree is often one of the factors that are required in the optimal problem solution. This requirement can be achieved in several ways:

- By penalizing the part of the individual fitness which contains a complex object tree,
- Method of targeted *shortening structural mutation* of individual,
- The direct shortening of the tree using algebraic adjustments - Algebraic Reducing Tree (ART).

The last-mentioned method can be realized by the TE, where all of individuals do not contain the genotype, and then a change in the phenotype is not affected by treatment with genotype. The realization of above mentioned problem with individual, which use genotype would be in this case very difficult. This new method is based on the algebraic arrangement of the tree features that are intended to reduce the number of functional blocks in the body of individuals (such as repeating blocks "unary minus", etc.).

In view of the object tree complexity of the individual and also for subsequent crossover is preferable to have a function in the form $x = 3 \times a$, than $x = a + a + a$, or more generally $x = n \times A$. Another example is the shortening of the function $x = -(-a)$, where the form $x = a$ is preferable (it is removing redundant marks in the object tree individual). The introduction of algebraic modifications of individual phenotype leads to the shorter result of the optimal solution and consequently to the shorter presentation of the individual and shortening the time of calculation of the function that is represented in object tree and also to find optimal solutions faster because of higher probability of crossover in the suitable points with higher probability to produce meaningful solutions. The essential difference stems from the use of direct contraction of trees, which leads to significantly shorter resulting structure than without using this method.
The arithmetic tree reducing method is shown on Fig. 9. The tree on the beginning of the process of ART is shown under the number 1, the resulting tree after use ART is shown under number 4.
The ART method uses the dictionary by which searches in the trees of individuals the subtrees contained in the dictionary, that reduces to the shape defined in the dictionary. The dictionary of ART method is shown on Fig. 10.

4. Fitness criterions

Considering to uses the transplant evolution for optimizing of general controller, the fitness function was experimentally defined as multi-criteria function of:

- Integral criterions: $I = \int_0^\infty e(t)dt$, $I = \int_0^\infty |e(t)|dt$, $I = \int_0^\infty [e(t)]^2 dt$, $I_{ITAE} = \int_0^\infty t|e(t)|dt$, $I_{ITSE} = \int_0^\infty t[e(t)]^2 dt$
- The count of extremes of controlled variable
- Average control error at the end of desired value interval
- A ratio of the number of points in the control error tolerance to the total number of points

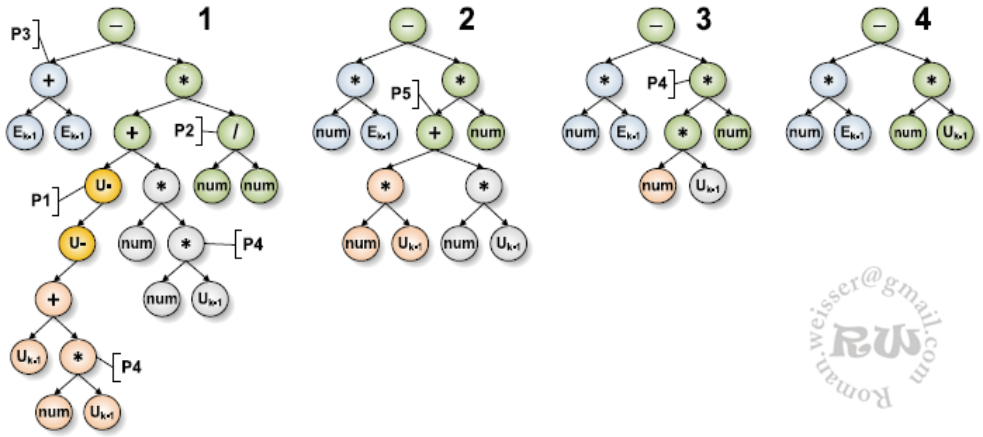

Fig. 9. Arithmetic tree reducing

$f_{2s}, f_{2m}, a, a_1, a_2$ Substitutional symbols of terminals

$f_{2s} \in \Sigma^{2s}$

$f_{2m} \in \Sigma^{2m}$

$\Sigma^{2s} \in \{+,-\}$

$\Sigma^{2m} \in \{*,/\}$

$(a, a_1, a_2) \in \Sigma^{rv}$

$a = a_1 = a_2$

$\Sigma^{re} = \Sigma^{2s} \cup \Sigma^{2m} \cup \Sigma^{rv}$

$e \in \Sigma^{re}$

$pair(a) = f_{2m}(num, a) \mid f_{2m}(a, num)$

P1	U+(e)	::=	e
	U-(U-(e))	::=	e
	U-(U+(e)) | U+(U-(e))	::=	U-(e)
P2	$f_{2m}(num, num) \mid f_{2s}(num, num)$	::=	num
P3	$+(a_1, a_2)$	::=	$*(num, a)$
	$-(a_1, a_2)$	::=	num
	$/(a_1, a_2)$	::=	num
P4	$f_{2m}(num, pair(a)) \mid f_{2m}(pair(a), num)$	::=	$*(num, a)$
	$f_{2s}(num, pair(a)) \mid f_{2s}(pair(a), num)$	::=	$*(num, a)$
P5	$f_{2s}(pair(a), pair(a))$	::=	$*(num, a)$

Fig. 10. Dictionary of arithmetic tree reducing method (ART)

- A ratio of the length of the curve of controlled variable to length of the curve of desired variable
- Maximal absolute overshoot of controlled variable

Maximal absolute overshoot of manipulated variable

The target of optimization is a minimize ITAE, minimize count of extremes of controlled variable, maximize the number of points within the control error tolerance, minimize of maximal overshoot of manipulated variable and controlled variable.

The fitness criterions is shown on Fig. 11.

5. Evaluation of simulation process in the case of error during simulation

In order to evaluate the simulation control according to the criteria set out in chapter 0, it is necessary to complete the whole process simulation. If the simulation process will not be completed for any reason (for example: divide by zero, ± ∞, etc.), the algorithm must be able to evaluate the simulation by another way. These criteria will vary depending on the optimized problem. In the event of PSD controller optimization, the fitness is evaluated according to the calculated number of cycles of recurrent equations PSD controller. More precisely, the ratio C_C / C_{CP}, where C_C is actually calculated number of cycles and the C_{CP} is the expected number of cycles.

Example: simulation time is *10s*, sampling period is *0.1s*. It follows that the total expected number of calculations of the recurrent equation PSD controller is *100*. If the simulation will be interrupted for any reason during the calculation of recurrent equations at *8 seconds*, the ratio C_C / C_{CP} = 79 = 100 = 0.79 (79 successfully completed the calculations).

If the optimization of recurrent equation of general controller is started, the fitness is evaluated according to successfully evaluated nodes in tree structure of individual during the simulation of the control. This is described by ratio (C_N/C_{NC}), where C_N is count of successfully evaluated nodes, C_{NC} is a number of all nodes in the structure of the recurrent equation of the controller.

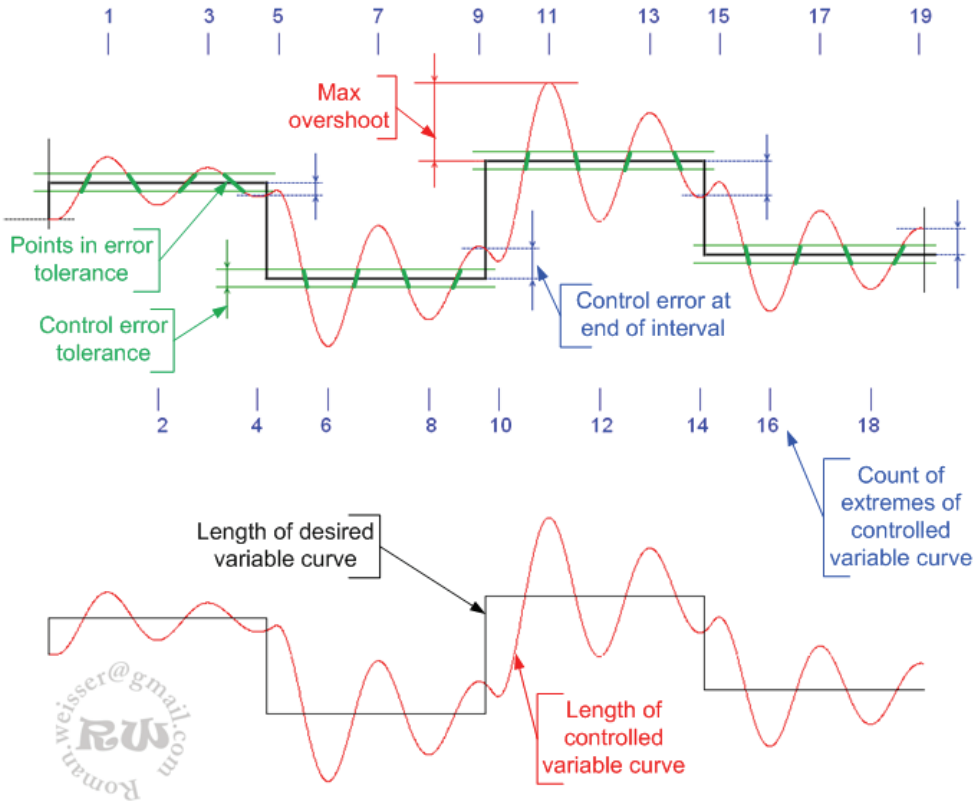

Fig. 11. Fitness criterions of control process

Example: simulation time is *10s*, sampling period is *0.1s*, the equation of the controller is in the form Uk = 3 * Ek. This equation shows that the function tree has 3 nodes (C_{NC} = 3). With each successive calculation of the recurrent equation will be calculated only three nodes. If the simulation will be interrupted for any reason during the calculation of recurrent equations at 8 seconds, when calculating the product of the nodes 3 and Ek is the resulting number of units calculated by 79 * 3 +1 * 2 (79 times was successfully completed the calculation of three nodes and in the latter calculation were calculated only 2 nodes). It means, the ratio is: CN/CNC = 239/3 = 79.66. In the case where the ratio is divided by length of simulation and size of sampling period, we receive same ratio as in the case optimization of PSD controller (79.66/100 = 0.7966). The counting of successfully calculated nodes of recurrent equation of general controllers is more precise. It can be seen in previous example. If the simulation is finished, it does not mean that fitness will be successfully evaluated. Whereas that the fitness function make multi-criterion fitness evaluation of simulation, it can occur ± ∞. In this case the fitness is evaluated by successfully calculated number of criterions. It means that evaluation of simulation can be done by three ways:

1. The simulation was not successfully finished. In this case the fitness is calculated from the ratio C_C/C_{CP}, or C_N/C_{NC}.

2. The simulation was successfully finished. In this case the fitness is calculated from successfully finished criterions.
3. The simulation and evaluation of fitness were successfully finished.

It means that way of evaluation 3 is every better than evaluation 2. Way of Evaluation 2 is every better than evaluation 1 ($fi_3 < fi_2 < fi_1$).
The evaluation of successful calculated fitness criterions are not shown in the picture Fig. 12. This type of evaluation is very important during optimization of the general controller. If those criterions would not be added, the optimal solution could not be found. In some case the successful solution was found after 3000 generations of DE!

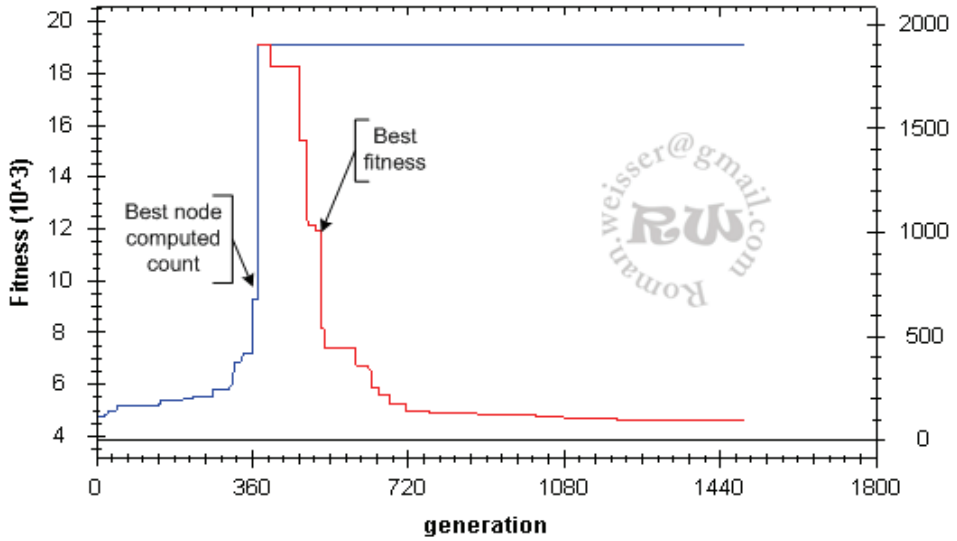

Fig. 12. Two types of fitness evaluation

6. Results

We tested the TLTE method for optimization of recurrent equation of general controllers. There is some results of optimization for two following systems.

6.1 Integral system with transport delay

$$G_S(S) = \frac{5}{s(9s+1)} e^{-2s} \tag{2}$$

In Fig. 13 we compare 3 types of controllers. There is one PSD controller marked PSD_DE and two general controllers marked General_DE and General_TLTE. The curve marked PSD_DE is PSD controller. Parameters (Kr, Ti, Td) of this controller were optimized by Differential Evolution (DE). The curve marked General_DE is for general controller which has the control equation in PSD equation form, but parameters q0, q1, q2 were optimized directly by DE. The relation between q0, q1, q2 and Kr, Ti, Td are shown in Table 7. The

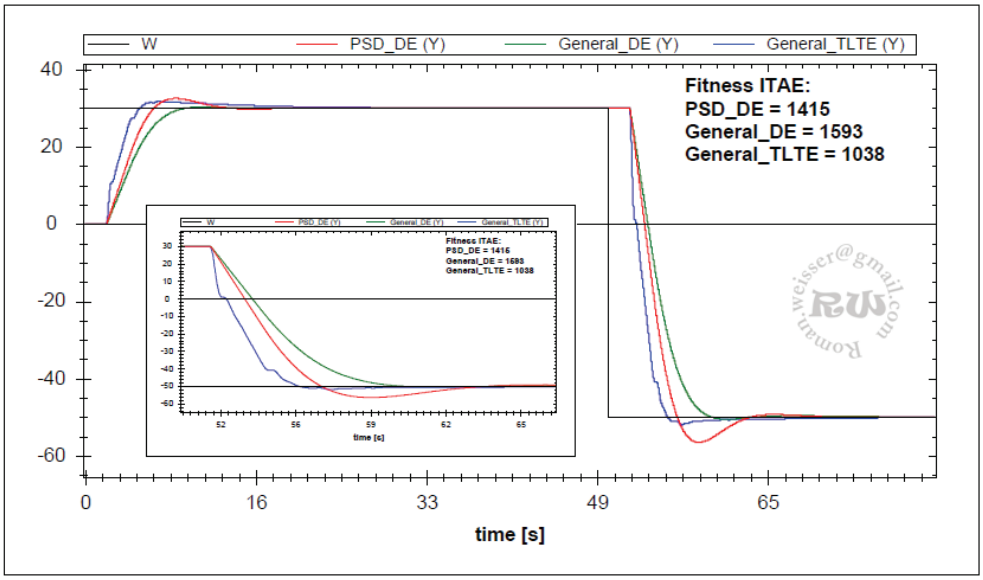

Fig. 13. Step response for integration system with time delay

curve marked General_TLTE is for general controller with general control equation that was optimized by Two-Level Transplant Evolution (TLTE). As you can see, the best result gives the General_TLTE. In this case we receive the recurrent control equation with following form:

```
Uk = (Ek_5+(((-((Ek_5*(-4,69210604912152))-(-((((Yk*5,14847786853854)*
     Yk_3*1,59643919322167))+((Ek_5+22,2867094434306)*Ek_2))*
     (Uk_2*(-3,80995100289523E-06))))))+((11,9436857350138*Ek)+
     (((-0,322492294234783)*Uk_3)+((-18,5463669625012)*Ek_2))))+
     (Ek_1+(-0,00399257533723234))))
```

	q_0	q_1	q_2
ZOBD	$k_R\left(1+\frac{T_D}{T}+\frac{T}{T_I}\right)$	$-k_R\left(1+2\frac{T_D}{T}\right)$	$k_R\frac{T_D}{T}$
DOBD	$k_R\left(1+\frac{T_D}{T}\right)$	$-k_R\left(1+2\frac{T_D}{T}-\frac{T}{T_I}\right)$	$k_R\frac{T_D}{T}$
LICHO	$k_R\left(1+\frac{T_D}{T}+\frac{T}{2T_I}\right)$	$-k_R\left(1+2\frac{T_D}{T}-\frac{T}{2T_I}\right)$	$k_R\frac{T_D}{T}$

Table 7. Examples of calculation of reccurent equation parameteres (q0, q1, q2) or (Kr, Ti, Td)

6.2 Second order system with time delay

$$G_S(S) = \frac{k_S}{T_0^2 s^2 + 2\xi T_0 s + 1} e^{-T_d S} \tag{3}$$

where $k_s = 1$, $T_0 = 1$, $\xi = 0.1$, $T_d = 1$

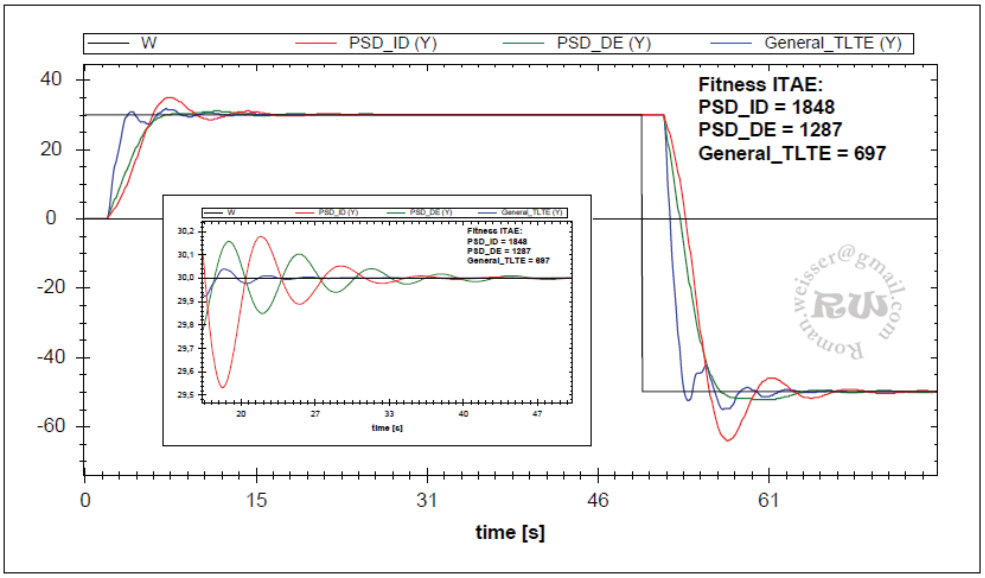

Fig. 14. Step response for oscilating proportionate second order system system with time delay

In Fig. 14 we compare 3 types of controllers. There is two types of PSD controllers marked PSD_ID, PSD_DE, and one general controllers marked General_TLTE. The curve marked PSD_ID is for PSD controller that was set by method inverse dynamics.
As you can see, the best result gives again the General_TLTE. In this case we receive the recurrent control equation with following form:

```
Uk=(((Yk_3*15,3645727059621)+(((Ek*3,80273057221998)-
    ((Ek-0,00031083646885983)+((Uk_5*0,571665466205751)+
    (Yk_4*13,7928925600503))))-(Ek_3*0,958174106578814)))-Ek_4)
```

7. Conclusion

The Two-Level Transplant Evolution (TLTE) was successfully use for automatic generation of control programs of general controllers. We tested this algorithm on many problems, only two examples were described in this paper. We hope that this new method of controller design will be use in practice, not only for simulation.

8. Acknowledgement

This work has been supported by Czech Ministry of Education No: MSM 00216305529 Intelligent Systems in Automation and GA ČR No: 102/09/1668.

9. References

BACKUS, J. W., et al. ALGOL-like Languages. Volume 1. Cambridge, MA, USA: Birkhauser Boston Inc., 1997. Revised report on the algorithmic language ALGOL 60, p. 19-49.

Koza J.R., Genetic Programming: On the Programming of Computers by Means of Natural Selection, The MIT Press, 1992

Kratochvíl O., Ošmera P., Popelka O. 2009: Parallel grammatical evolution for circuit optimization, in Proc. WCECS, World Congress on Engineering and Computer Science, San Francisco, 1032-1040.

Li Z. and Halang W. A. and Chen G. 2006: Integration of Fuzzy Logic and Chaos Theory; paragraph: Osmera P.: Evolution of Complexity, Springer, 527 – 578.

O'Neill, M., Brabazon, A.: Grammatical Differential Evolution, In Proc. International Conference on Artificial Intelligence (ICAI'06) CSEA Press Las Vegas, Nevada.

O'Neill M. and Ryan C. 2003: Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language Kluwer Academic Publishers.

Ošmera P. and Šimoník I. and Roupec J. 1995: Multilevel distributed genetic algorithms. In Proceedings of the International Conference IEE/IEEE on Genetic Algorithms, Sheffield, 505–510.

Ošmera P. and Roupec J. 2000: Limited Lifetime Genetic Algorithms in Comparison with Sexual Reproduction Based GAs, Proceedings of MENDEL, Brno, Czech Republic, 118 – 126.

Popelka, O.: Parallel Grammatical Evolution for Circuit Optimization. In: Proceedings of MENDEL 2007, Prague, Czech Republic, pp. 88–92 (2007)

Price K. Differential evolution: a fast and simple numerical optimizer, Biennial Conference of the North American Fuzzy Information Processing Society, 1996, NAFIPS, IEEE Press, New York, NY, 524-527.

Rukovanský I., Evolution of Complex Systems. 8th Joint Conference on Information Sciences. Salt Lake City. Utah. July 21-25, 2005.

Rukovanský I. Optimization of the throughput of Computer Network Based on Parallel EA. In Proceedings of the World Congress on Engineering and Computer Science WCECS 2009, San Francisco, CA, Oct. 20-22, 2009, Vol. II, pp. 1038-1043

Rukovanský I., Popelka O. Increasing Performance of Computer Networks Via Node to Node throughput Optimization. In : Proceedings of the Information Conference on Soft Computing Applied in Computer and Economic Environments, ICSC 2008, Kunovice : EPI, Ltd. Czech Republic, January 25, 2008, pp. 171-175

Weisser R., Ošmera P., Matoušek R., Transplant Evolution with Modified Schema of Differential Evolution : Optimization Structure of Controllers. In International Conference on Soft Computing MENDEL. Brno : MENDEL, 2010.

Weisser R., Ošmera P., Šeda, M., Kratochvíl, O. Transplant Evolution for Optimization of General Controllers. In European Conference on Modelling and Simulation. 24th. Kuala Lumpur (Malaysia) : ECMS 2010. s. 250 -- 260.

Weisser R., Ošmera P., Two-level Tranpslant Evolution, In East West Fuzzy Colloquium, 17th Zittau Fuzzy Colloquium 2010

Weisser R., Ošmera P., Two-level Tranpslant Evolution: Optimization of General Controllers, In East West Fuzzy Colloquium, 17th Zittau Fuzzy Colloquium 2010

Zelinka I., Oplatkova Z. Nolle L., Analytic Programming - Symbolic Regression by Means of Arbitrary Evolutionary Algorithms, In: Special Issue on Inteligent Systems, International Journal of Simulation, Systems, Science and Technology, Volume 6, Issue 9, August 2005, pp 44 - 56, ISSN 1473-8031