5

Two-Level Transplant Evolution for Optimization of General Controllers

Roman Weisser, Pavel Ošmera, Jan Roupec, and Radomil Matousek
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 the optimization of control equations of general controllers it is suitable to combine two evolutionary algorithms. The main goal in the first level of TLTE is the optimization of the structure of general controllers. In the second level of TLTE the concrete parameters are optimized and the unknown abstract parameters in the structure of equations are set. The method TLTE was created by the combination of the Transplant Evolution method (TE) and the Differential Evolution method (DE). The Transplant Evolution (TE) optimizes the structure of the 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 to find directly in TE without DE. Some new methods for evaluation of the quality of the found control equation are described here, which allow us evaluate their quality. These can be used in the case when the simulation of the control process cannot be finished. Some practical applications are shown in the results. In all calculation of TLTE the control equation had a better quality of the control process, than the classical PSD controllers and Takahashi`s modification of the PSD controller.

2. Transplant Evolution

Transplant Evolution (TE) was inspired by biological transplantation. It is an analogy to the transplant surgery of organs. Every transplanted organ was created by DNA information but some parts of an ill body can be replaced by a new organ from the database of organs. The description parts of individual (organs) in the database are not stored on the level DNA (genotype). In Transplant Evolution (TE) every individual part (organ) is created by the translation of grammar rules similar to Grammatical Evolution (GE), but Transplant Evolution (TE) does not store the genotype and the grammatical rules are chosen randomly. The newly created structure is stored only as a 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 the Differential Evolution algorithm (DE). Optimization of the numerical parameters of general controllers in recurrent control equations of general controllers is a very difficult problem. We applied the second level of optimization by the Differential Evolution method (DE). The individuals in TE and TLTE are represented only by a phenotype in the shape of an object tree. During the initialization of population and during the creation of these phenotypes, similar methods are used as in GE. In Grammatical Evolution the numerically represented chromosomes are used. The codons of these chromosomes are used for the selection of some rule from a set of rules. In Transplant Evolution the chromosomes and codons are not used, but for the selection of some rule from a set of rules randomly generated numbers are used. These numbers are not stored in the individual chromosome. The new individuals in the population are created with the use of analytic and generative grammar and by using crossover and mutation operators. These operators are projected for work with the phenotype of an individual, similarly as in GP. Because the individuals of TE and TLTE are represented only by phenotype, it was possible to implement these? advanced methods in the course of evolution:

- an effective change of the rules in the course of evolution, without the loss of generated solutions,
- a difference of probability in the selection of rules from the set of rules and the possibility of this 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: $U_{k-1} + E_k + E_{k-1} * num$),
- new methods of crossover are 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 the set of rules by the following equation:

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

Where: *random* is a pseudo-random number generator, *maxInt* is a high number, % is the remainder operator (modulus), and *rules_count* denotes the number of possible rules to transcribe a 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).The Original initialization algorithm GE uses forward processing of grammatical rules. In the Grammatical Evolution the method of crossover and mutation are made on the genotype level. The phenotype is created by a later translation of this genotype. This way of mutation and crossover does not allow the using of advanced method in crossover and mutation operators, which does not destruct the already created parts of the individual (O'Neill M. and Ryan C., 2003). During the evolution of this algorithm the Backward Processing of Rules (BPR) arose (Popelka, O., 2007). The BPR method uses gene marking. This approach has the hidden knowledge of tree structure. Due to the BPR the advanced and effective methods of crossover and mutation can be used. Anyhow the BPR method has some disadvantages. Due to these disadvantages the new

method Transplant Evolution (TE) was created. The TE method does not store genotype information. The equation 1 is used for the selection of a rule from rules base. The advantage of TE is the possibility to use both types of grammatical processing (forward and backward)

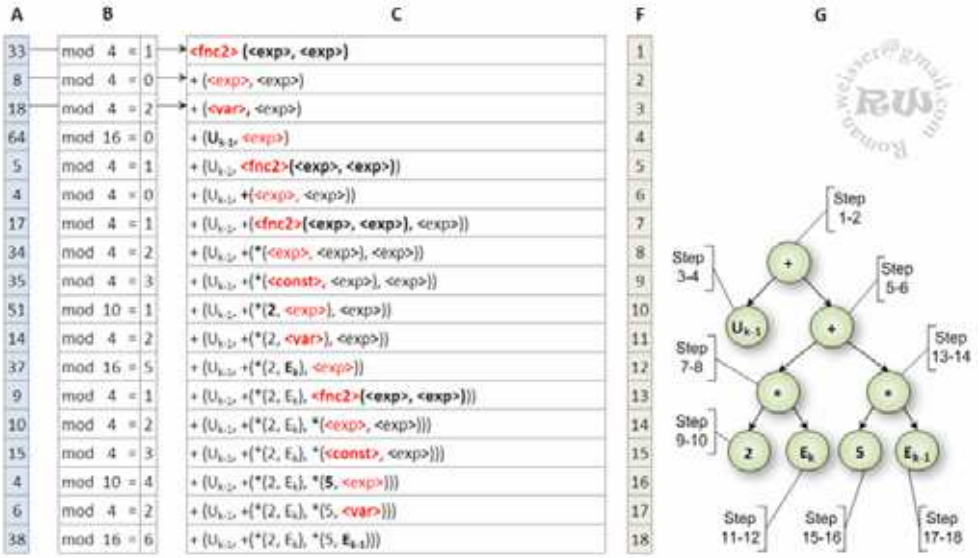


Fig. 1. Translation in TE (forward processing)

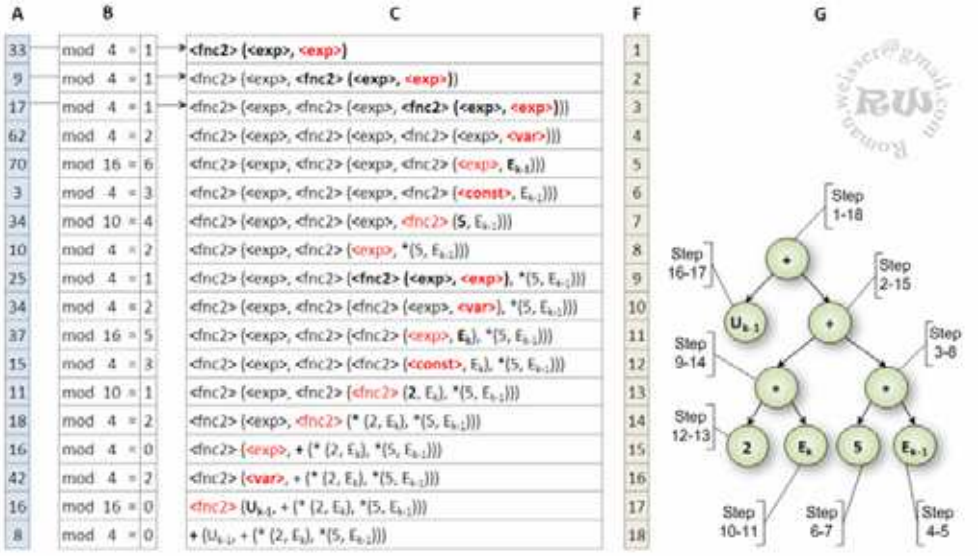


Fig. 2. Translation in TE (backward processing)

with the same results, because TE works only with the phenotype and the 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 $U_{k-1} + 2* E_k + 5 * E_{k-1}$.
In *column A* are shown the randomly generated values. These values are not stored anywhere! The values are generated only when is necessary to select from more grammatical rules. In *column B* is shown the arithmetic operation. In *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 the phenotype). In *column F* is the order of operations. These numbers are used for a description of the tree nodes initialization order. The tree initialization is shown at *column G*. Each node of the tree in *column G* is described by a number in the form *Step X-Y*. In the ***Step X-Y***, *X* represents the step in which the node was created. *Y* represents the 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 a different order of fully initialization. The generated numbers in column A were changed too.
Both the examples of initialization of the individual expected generative grammar rules (productive rules) are 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

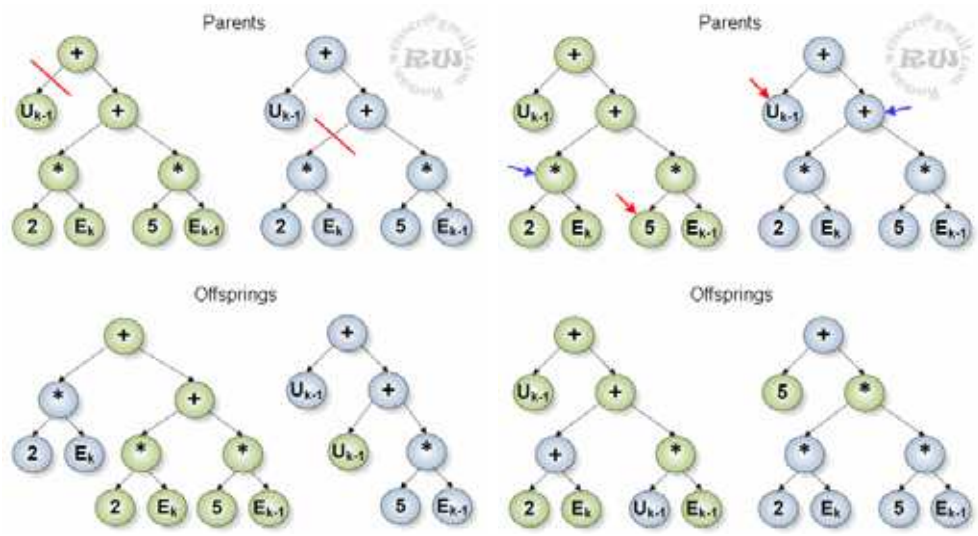
The main principle of the initialization of individual in Transplant Evolution is described by the recurrent pseudo-code in Fig. 3. In this pseudo-code it can be seen that some rules can be disabled or the probability of selection can be changed. This possibility can be seen on row 2 in Fig. 3. The random selection of rule from rule base are shown on row 9. This approach is realized by the method called Get_ random_rule. The method CreateSubTree on row 11 creates 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 in a 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 it* is neccessary to keep change of the same types of nodes. This request is possible to express by this relationship:

```
Input: G – context-free grammar; P – array of rules; L_max – max nodes count
Result: individual
if length[this] ≥ L_max then
    Disable_rules('function') ;
end
for i ← length[P] to i ≥ 1 do
    if P[i] is terminal symbol then
        rule ← P[i]
    else
        P_temp ← array of rules for the nonterminal P[i] in G indexed from zero;
        rule ← Get_random_rule(P_temp) ;
    end
    sub_tree ← CreateSubTree(G; rule; L_max) ;
    this ← add child sub_tree ;
end
if P[0] is terminal then
    this.terminal = P[0] ;
else
    sub_tree ← CreateSubTree(G; P[0]; L_max) ;
    this.terminal ← sub_tree;
end
return this;
```

Fig. 3. Pseudo-code of initialization of individuals



a) b)

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

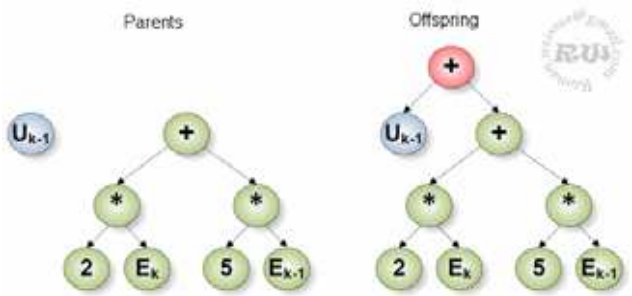


Fig. 5. Crossover by linking trees

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

The method of *crossover by linking trees* is a new method. This method creates one offspring from two parents. The parts of the parents are linked by the newly generated node. The newly generated node has two subparts. The first one of them is the tree of the first parent and the second one of them is the tree of the second parent.

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

The selected node in the tree G1 is marked by an orange color (or a different level of gray). The tree after mutation is marked G2 and the mutated node is marked an orange color (or different level of gray) too. The mutation was made by the 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. The modulus operations are shown in column B. The processing of rules is shown in column C.

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

Structural mutations, 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 the

defined grammar (see Fig. 7). Conversely the **shortening structural mutation** replaces a randomly selected node of the tree, including its child nodes, by a node which is described by a 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).

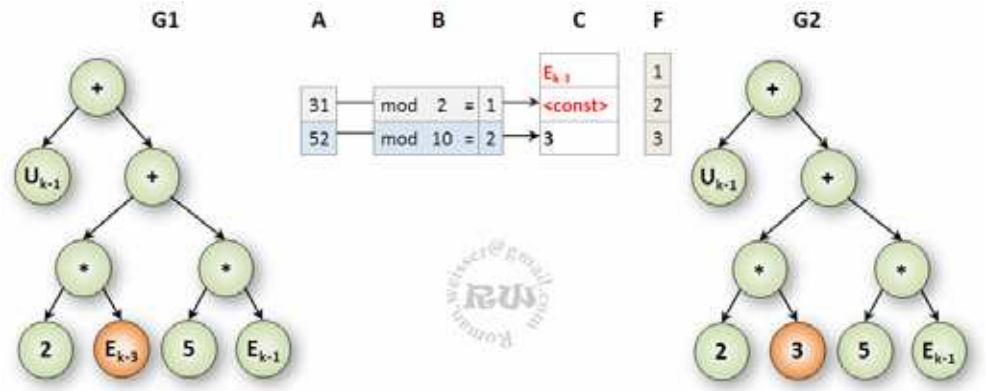


Fig. 6. Non-structural mutation

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

Table 2. Analytic grammar (non-structural mutation)

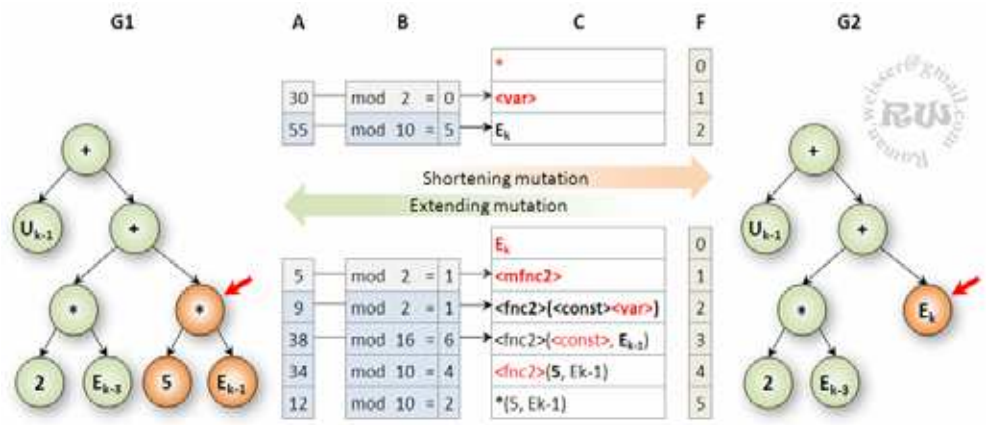


Fig. 7. Structural mutation (Shortening, Extending)

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

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

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

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 a 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. Joining the two evolution algorithms has the goal to eliminate the very hard optimization of the numerical parameters by the grammar based evolutionary algorithms. The DE method was chosen for high power of optimization of the real number parameters.

The generative grammar and analytic grammar rules are needed to be modified for the generation or mutation of abstract numerical parameters. To these grammar rules it 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 the implementation viewpoint, the joining of the two evolutionary algorithms is the most difficult process. For the best flexibility the interconnecting of three separate computational modules that are join by an interlayer were chosen. The interlayer mediates a communication among the modules and prepares data in the required form. With this implementation it is possible to use each module separately for another optimization problem, or similarly to change some module by another module.

3.1.1 Transplant Evolution Module

The transplant Evolution Module (MTE), see Fig. 8 ensures optimization of the structure of controllers. The principle of this algorithm was described in the 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 the two-level structure. Optimization of the parameters is provided by the Differential Evolution Module (MDE). The agent for the communication between these layers is the Interlayer, which is responsible for preparing the data in the desired form for each module.The input parameters for the MTE are the generative grammar rules P and the analytic grammar rules P^{-1}, which leads to the 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 it the vector of variables $\overline{num} = (num_1, ..., num_n)$ as one of the input parameters. The 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. The output parameter MDE is the optimized vector of parameters that is sent into the *Interlayer*.

3.1.3 Fitness Module

The fitness Module (FM), see Fig. 8, provides an evaluation of the models. The model is some structure of the 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 the fitness function after the finishing of the simulation. The evaluated model is send to the *Interlayer*.

3.1.4 Interlayer

The Interlayer, see Fig. 8 is the most important part in the whole architecture of Two-Level architecture. The Interlayer is the main module, which holds instances of all the computational modules. The *interlayer* provides communication among the computational modules through the communication interface. This module contains the knowledge about the strategy of optimization so this means what to be optimized and how to optimize it. The *interlayer* controls the parameters of all the 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 an Output Module (OM). The OM is a graphical output, which shows the evolution outputs, simulation of control outputs, etc. The most important task of the *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:

- To 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 the finishing of the optimization.
- To send a full model of the solution (structure + parameters) to the FM, during the request of the MTE for the calculation of the quality of an individual, which does not include any abstract parameter.

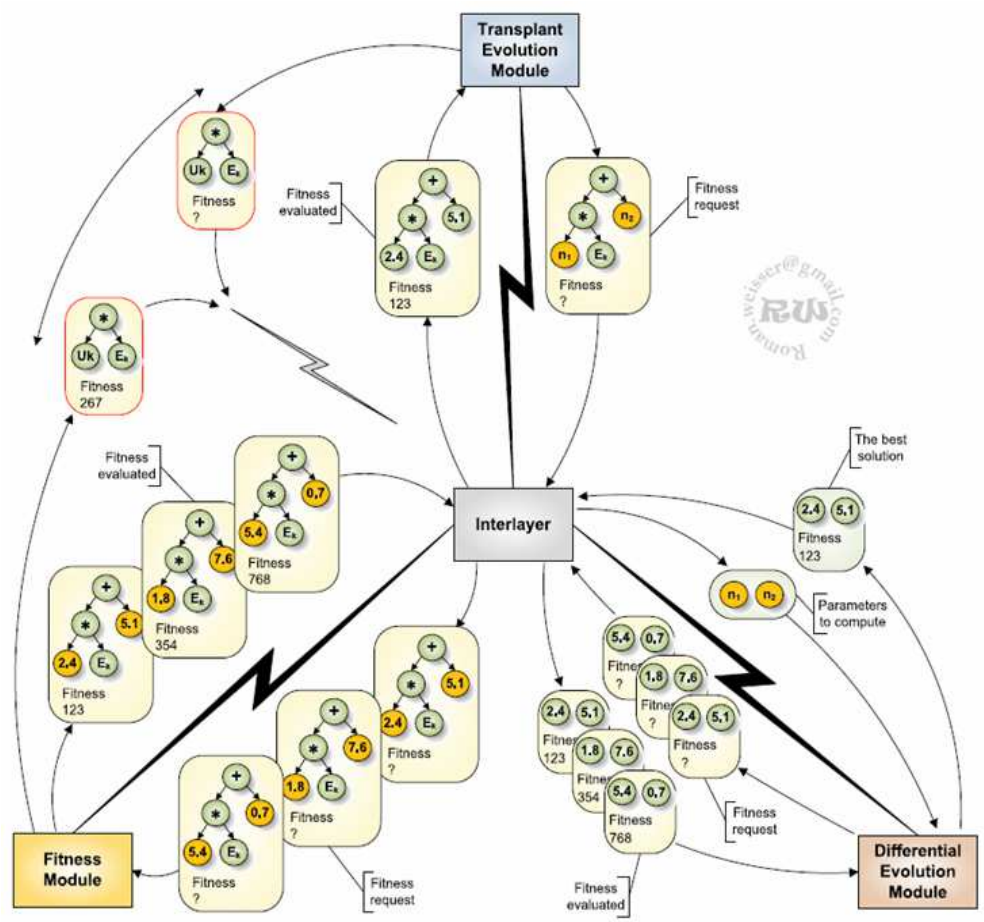


Fig. 8. Architecture and data flow in TLTE

3.2 Arithmetic tree reducing

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

- 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 the individuals do not contain the genotype, and then a change in the phenotype is not affected by treatment with the genotype. The realization of the above mentioned problem with an individual, which uses the 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 the subsequent crossover it is preferable to have a function in the form of $x = 3 \times a$, *rather* 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 the redundant marks in the object tree individual). The introduction of algebraic modifications of the individual phenotype leads to a shorter result of the optimal solution and consequently to a shorter presentation of the individual and the shortening of the time of calculation of the function that is represented in the object tree and also to finding optimal solutions faster because of the higher probability of crossover in the suitable points with a higher probability to produce meaningful solutions. The essential difference stems from the use of the direct contraction of trees, which leads to a 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 the use of ART is shown under number 4.

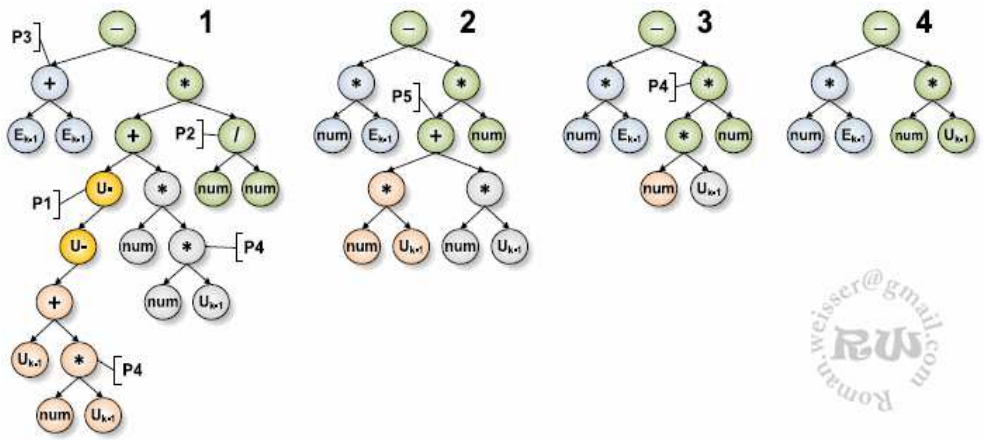


Fig. 9. Arithmetic tree reducing

$f_{2s}, f_{2m}, a, a_1, a_2$ 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)

The ART method uses a 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 the uses of the transplant evolution for the optimizing of the general controller, the fitness function was experimentally defined as the 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
- A ratio of the length of the curve of the controlled variable to the length of the curve of the desired variable
- Maximal absolute overshoot of controlled variable

Maximal absolute overshoot of manipulated variable

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

The fitness criterions are shown on Fig. 11.

5. Evaluation of the 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 the recurrent equations of the PSD controller. More precisely, the ratio C_C / C_{CP}, where C_C is the actual 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 calculations).
If the optimization of the recurrent equation of the general controller is started, the fitness is evaluated according to successfully evaluated nodes in the tree structure of the individual during the simulation of the control. This is described by the ratio (C_N/C_{NC}), where C_N is the count of successfully evaluated nodes, and C_{NC} is a number of all the nodes in the structure of the recurrent equation of the controller.

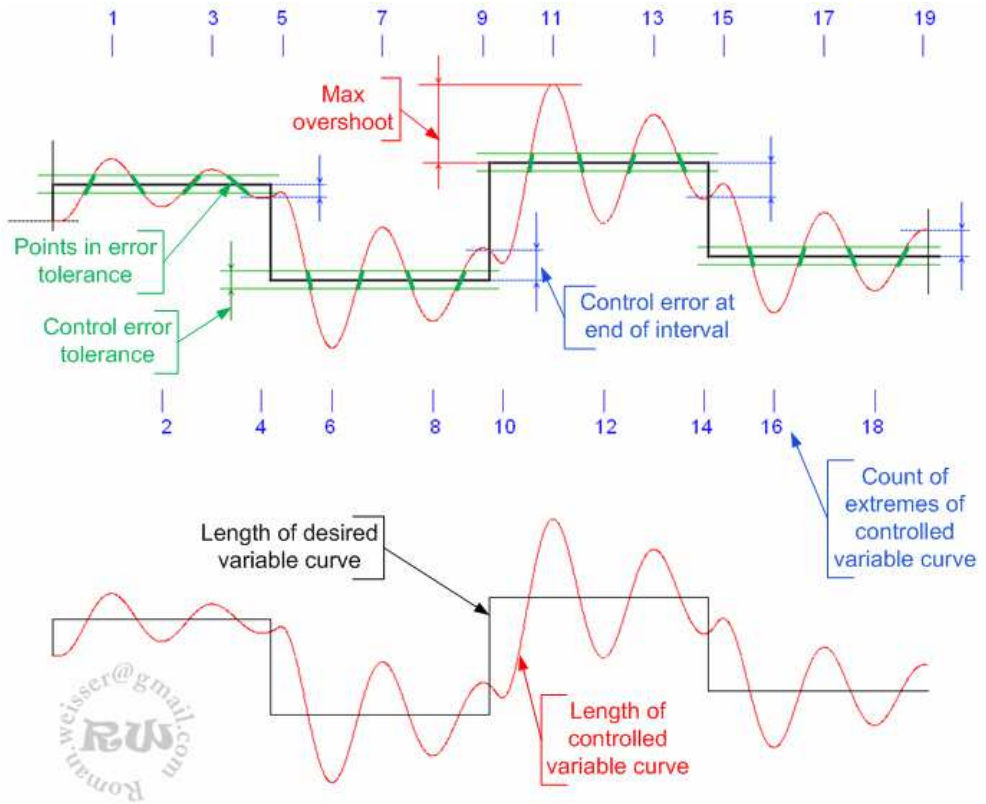


Fig. 11. Fitness criterions of control process

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 only three nodes will be calculated. If the simulation will be interrupted for any reason during the calculation of the recurrent equations at 8 seconds, when calculating the product of the nodes 3 and Ek the resulting number of units is 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). This means, the ratio is: CN/CNC = 239/3 = 79.66. In the case where the ratio is divided by the length of simulation and the size of the sampling period, we receive the same ratio as in the case of 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 the previous example. If the simulation is finished, it does not mean that the 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 the successfully calculated number of criterions. It means that the 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 the successfully finished criterions.
3. The simulation and evaluation of fitness were successfully finished.

This means that way of evaluation 3 is every time better than evaluation 2, and the way of Evaluation 2 is every time better than evaluation 1 ($fi_3 < fi_2 < fi_1$).

The evaluation of the successfully calculated fitness criterions are now shown in the picture Fig. 12. This type of evaluation is very important during the optimization of the general controller. If those criterions would not be added, the optimal solution could not be found. In some cases the successful solution was found after 3000 generations of DE!

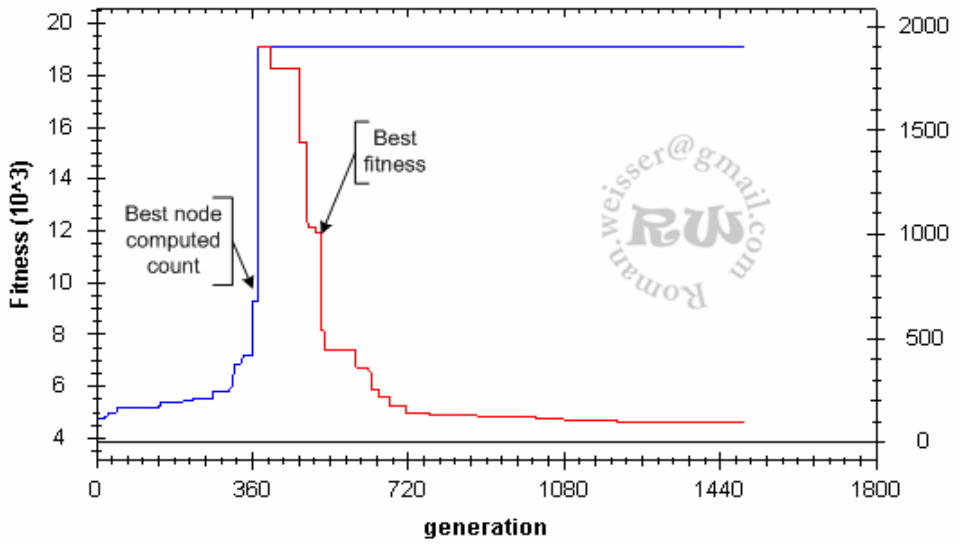


Fig. 12. Two types of fitness evaluation

6. Results

We tested the TLTE method for the optimization of the recurrent equation of general controllers. Here there are some results of the optimization for following two systems.

6.1 Integral system with a 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 the PSD controller. The parameters (Kr, Ti, Td) of this controller were optimized by Differential Evolution (DE). The curve marked General DE is for the general controller which has the control equation in PSD equation form, but the 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 curve marked General_TLTE is for the general controller with a general control equation that was optimized by Two-Level Transplant Evolution (TLTE). As you can see, the best result is given by the General_TLTE. In this case we receive the recurrent control equation with the 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))))
```

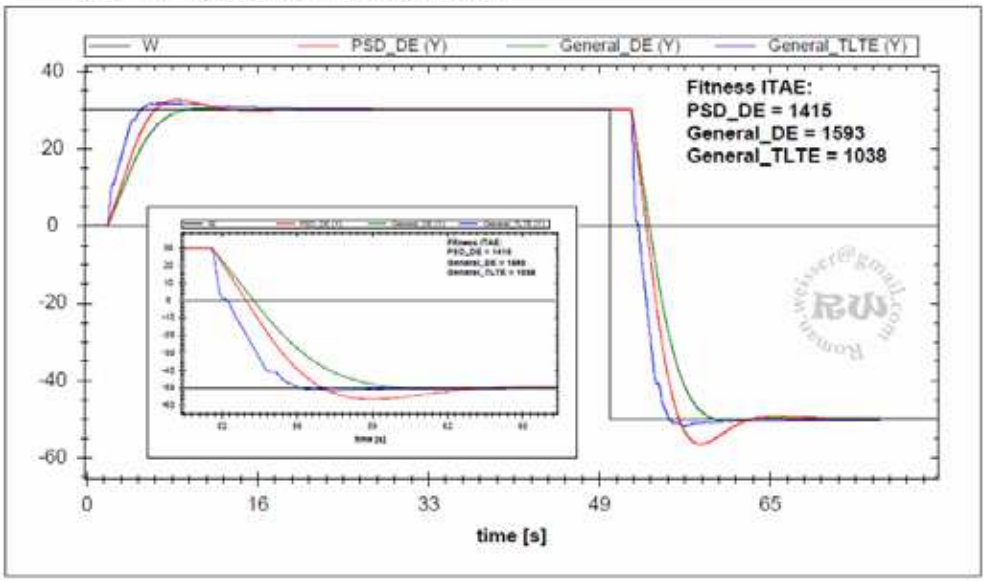


Fig. 13. Step response for integration system with time delay

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

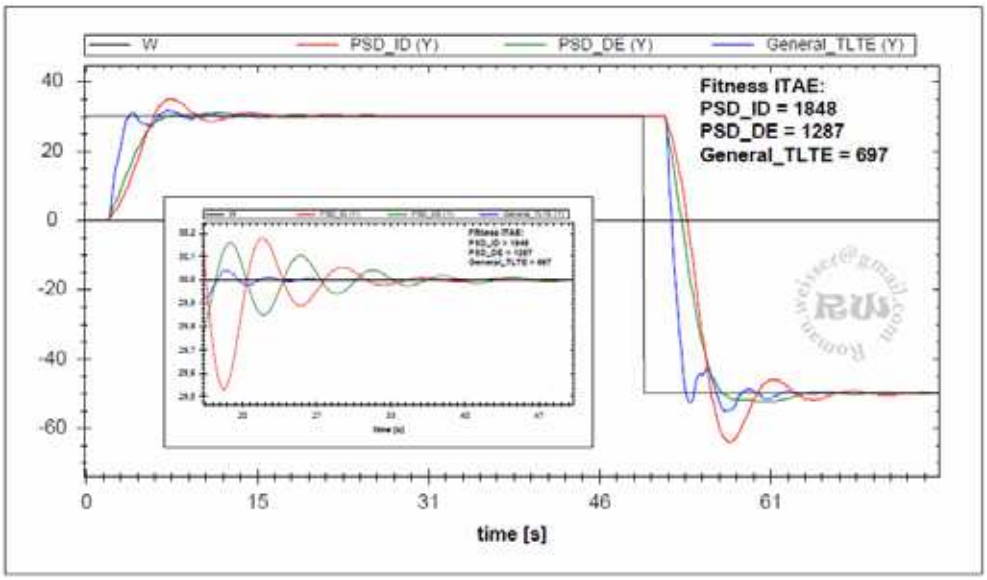


Fig. 14. Step response for oscilating proportionate second order system system with time delay

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

```
Uk=(((Yk_3*15,3645727059621)+(((Ek*3,80273057221998)-
   ((Ek-0,000331083646885983)+((Uk_5*0,571665466205751)+
   (Yk_4*13,7928925600503))))-(Ek_3*0,958174106578814)))-Ek_4)
```

7. Conclusion

The Two-Level Transplant Evolution (TLTE) was successfully used for the 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 used in practice, not only for simulation.

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

8. 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

New Trends in Technologies: Control, Management, Computational Intelligence and Network Systems
Edited by Meng Joo Er

ISBN 978-953-307-213-5
Hard cover, 438 pages
Publisher Sciyo
Published online 02, November, 2010
Published in print edition November, 2010

The grandest accomplishments of engineering took place in the twentieth century. The widespread development and distribution of electricity and clean water, automobiles and airplanes, radio and television, spacecraft and lasers, antibiotics and medical imaging, computers and the Internet are just some of the highlights from a century in which engineering revolutionized and improved virtually every aspect of human life. In this book, the authors provide a glimpse of the new trends of technologies pertaining to control, management, computational intelligence and network systems.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821