

Two-level Transplant Evolution for Optimization of General Controllers

P. Osmera, R. Weisser, M. Seda, and J. Roupec

Abstract—This paper describes the application of Two-Level Transplant Evolution (TE) that can evolve control programs using a variable length linear genome to govern the mapping of a Backus Naur Form grammar definition. TE combines Grammatical Evolution (on the genotype level) with Genetic Programming (tree structures on the phenotype level). To increase the efficiency of Transplant Evolution (TE) the parallel Differential Evolution was added.

Keywords—general controllers, optimization, two-level transplant evolution.

I. INTRODUCTION

THE aim of this paper is to describe a new optimization method that can create control equations of general controllers. 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) [8]. 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.

II. 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], [5], [9], [10]. 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

P. Osmera is with the European Polytechnical Institute Kunovice, Kunovice, Czech Republic (e-mail: osmera@fme.vutbr.cz).

R. Weisser was with Brno University of technology, Brno, Czech Republic He is now with seznam.cz (e-mail: roman.weisser@gmail.com).

M. Seda is with Brno University of Technology, Institute of Automation and Computer Science, Brno, Czech Republic (e-mail: seda@fme.vutbr.cz).

J. Roupec is with Brno University of Technology, Institute of Automation and Computer Science, Brno, Czech Republic (e-mail: roupec@fme.vutbr.cz).

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: $U_{k-1} + E_k + E_{k-1} * \text{num}$),
- new methods of crossover are possible to use, (for example crossover by linking trees)
- etc.

A. 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:

$$\text{rule} = \text{random}(0, \text{maxInt}) \% \text{rules_count} \quad (1)$$

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 (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 [3]. During the evolution of this algorithm the Backward Processing of Rules (BPR) [7] arose [11]. 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) 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 gene 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.

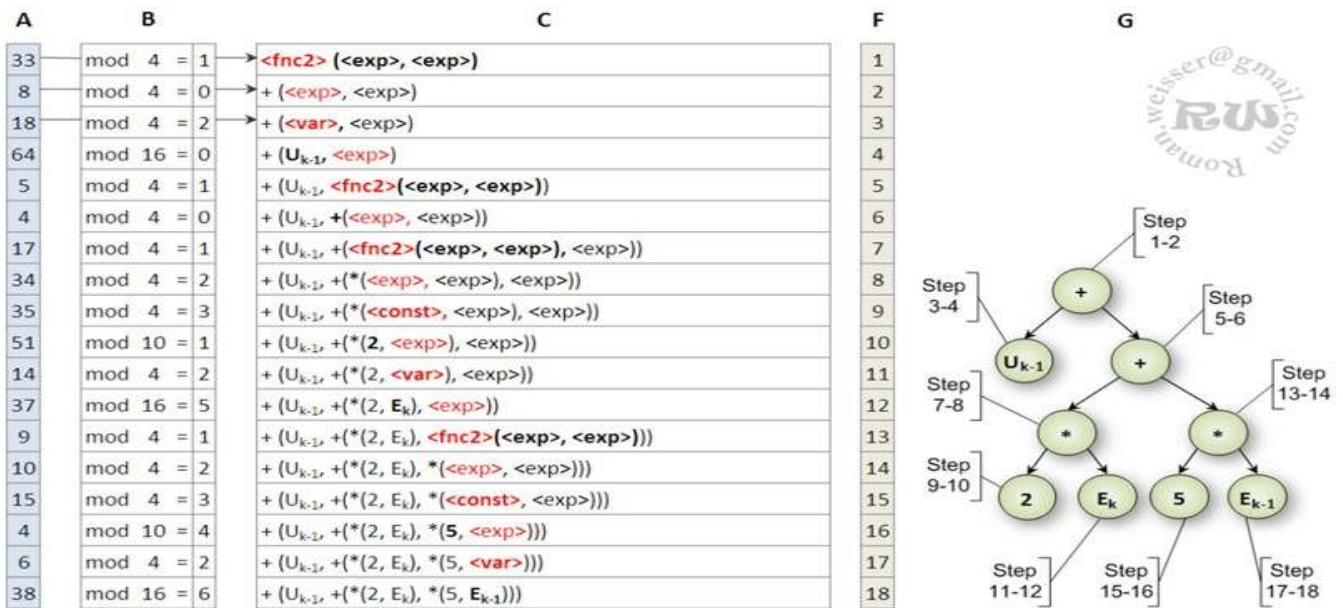


Fig. 1. Translation in GE (forward processing) [6]

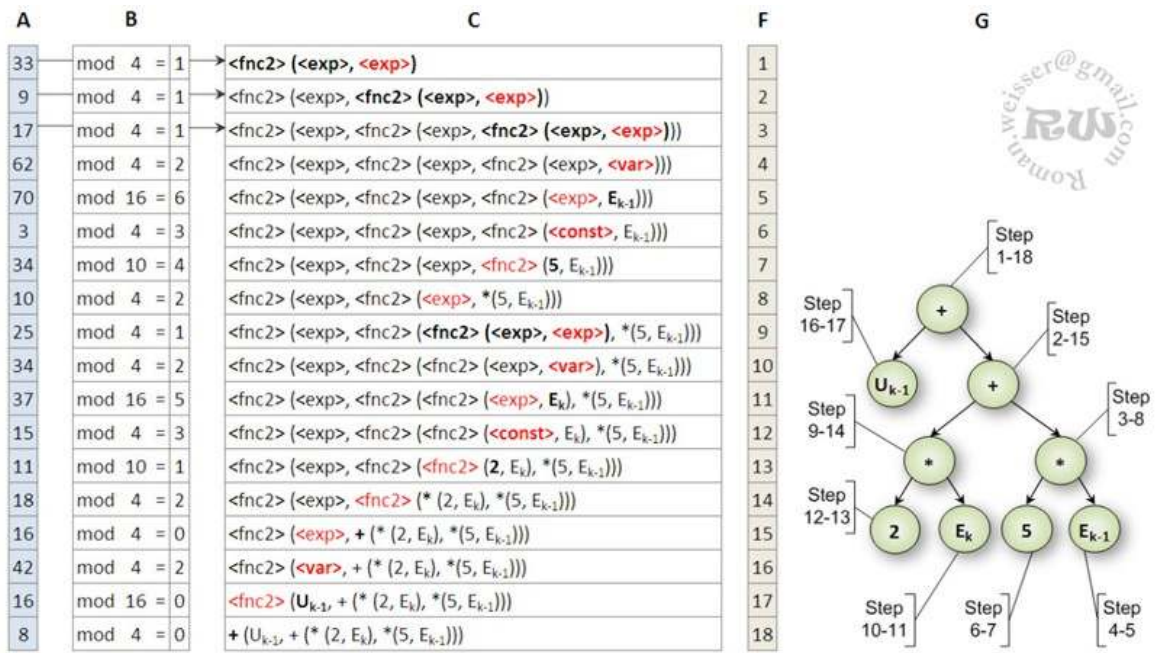


Fig. 2. Translation in GE (backward processing) [7]

Both the examples of initialization of the individual [2]. expected generative grammar rules (productive rules) are in

The TE uses three types of crossover. The first type of

TABLE 1. Generative grammar

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

prefix form. The generative grammar is defined in TABLE 1. The rules in this table are written in Backus Naur Form (BNF) [1]. 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.

B. 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)

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 necessary to keep change of the same types of nodes. This request is possible to express by this relationship:

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

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.

```

Input:  $G$  – context-free grammar;  $P$  – array of rules;  $L_{max}$  – max nodes count
Result: individual
1 If  $length(P) \geq L_{max}$  then
2   Disable_rules('function');
3 end
4 for  $i = length(P)$  to 1  $\geq i$  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

```

Fig.3. Pseudo-code of initialization of individuals

C. 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 (E_k , etc.), etc. See Fig.5.

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.

TABLE 2. Analytic grammar (non-structural mutation)

$U_{k-1} U_{k-2} U_{k-3} U_{k-4} U_{k-5} ::= \langle var \rangle \langle const \rangle$
$E_k E_{k-1} E_{k-2} E_{k-3} E_{k-4} E_{k-5} ::= \langle var \rangle \langle const \rangle$
$dE_k dE_{k-1} dE_{k-2} dE_{k-3} dE_{k-4} ::= \langle var \rangle \langle const \rangle$
$decimal ::= \langle var \rangle \langle const \rangle$
$U+ U- ::= \langle func1 \rangle$
$+ - * / ::= \langle func2 \rangle$

TABLE 3. Analytic grammar (extending structural mutation)

$U_{k-1} U_{k-2} U_{k-3} U_{k-4} U_{k-5} ::= \langle mfunc1 \rangle \langle mfunc2 \rangle$
$E_k E_{k-1} E_{k-2} E_{k-3} E_{k-4} E_{k-5} ::= \langle mfunc1 \rangle \langle mfunc2 \rangle$
$dE_k dE_{k-1} dE_{k-2} dE_{k-3} dE_{k-4} ::= \langle mfunc1 \rangle \langle mfunc2 \rangle$
$U+ U- ::= \langle mfunc2 \rangle$

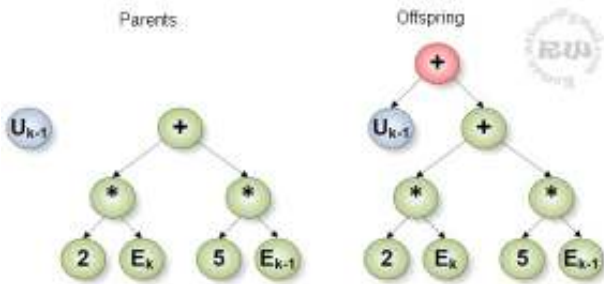


Fig. 4. Crossover by linking trees

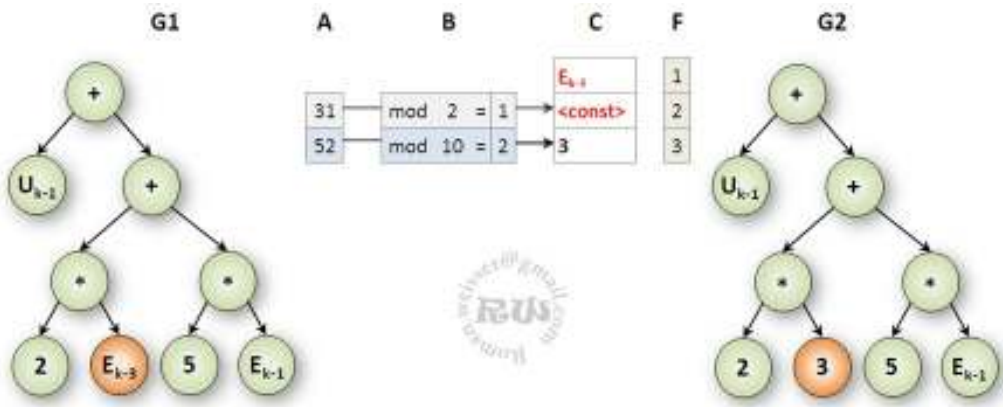


Fig.5. Non-structural mutation

TABLE 4. Analytic grammar (shortening structural mutation)

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

TABLE 5. Generative grammar (expansion for analytic grammar)

$\langle mfunc1 \rangle$	$::=$	$\langle func1 \rangle \langle var \rangle$	$ $	$\langle func1 \rangle \langle const \rangle$	\dots
$\langle mfunc2 \rangle$	$::=$	$\langle func2 \rangle \langle var \rangle \langle var \rangle$	$ $	$\langle func2 \rangle \langle const \rangle \langle var \rangle$	

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 [11, [12]. 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 [12].

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 Table 5 assumes the analytic grammar and generative grammar rules from tables Table 3, Table 4, and Table 1.

III. 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.

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

$\langle var \rangle$	$::=$	num	$\langle var \rangle$	$::=$	num
-----------------------	-------	-----	-----------------------	-------	-----

A. 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.

B. Transplant Evolution Module

The transplant Evolution Module (MTE) (see Fig.6) 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.

C. Differential Evolution Module

The Differential Evolution Module (MDE), see Fig. 6 ensures the optimization of abstract parameters in the structure of the individual. The principle function of *Differential Evolution* (DE) is explained for example in [8]. This module is activated by the *Interlayer*, which gives it the vector of variables 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*.

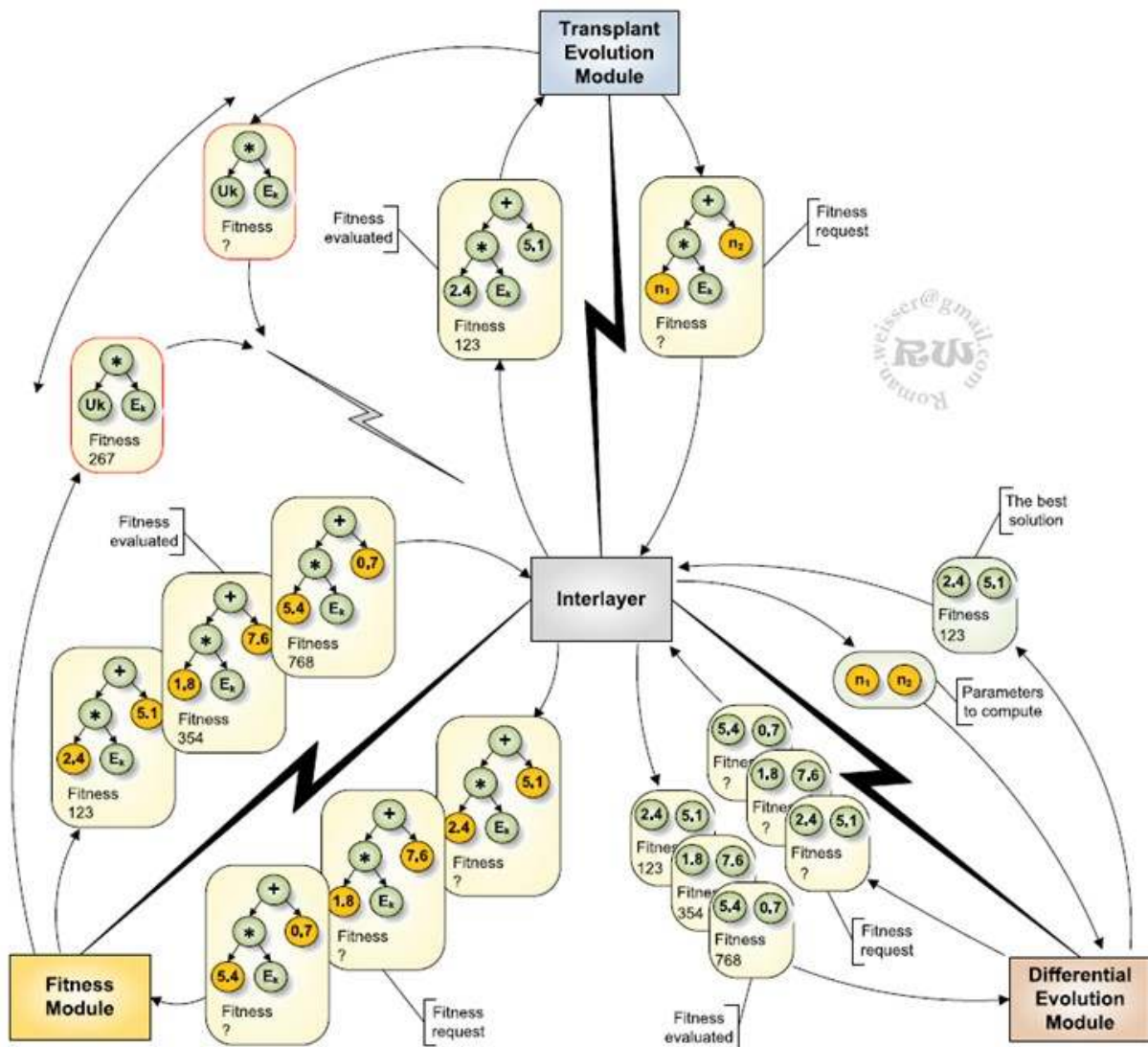


Fig. 6. Architecture and data flow in TLTE

D. Fitness Module

The fitness Module (FM), see Fig. , 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*.

E. Interlayer

The Interlayer, see Fig. 6 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. 6. 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,

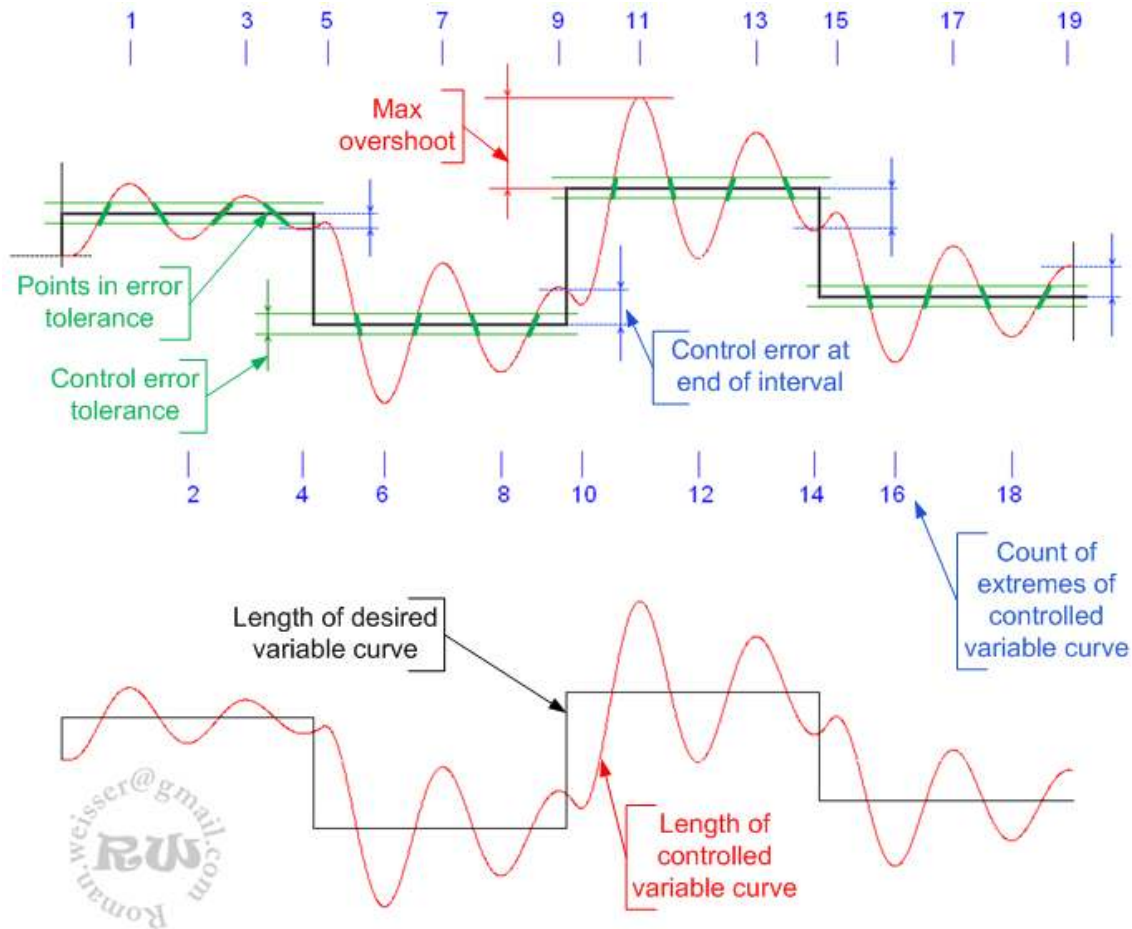


Fig. 7. Fitness criteria of control process

- which does not include any abstract parameter.

The fitness criteria are shown on Fig. 7.

IV. 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: $I = \int_0^{\infty} e(t)dt$, $I = \int_0^{\infty} |e(t)|dt$,
 $I = \int_0^{\infty} [e(t)]^2 dt$,
 $I_{ITAE} = \int_0^{\infty} t|e(t)|dt$, $I_{ITSE} = \int_0^{\infty} t[e(t)]^2 dt$ (3)

- 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.

V. RESULTS

We tested the TLTE method for the optimization of the recurrent equation of general controllers [11], [12]. Here there are some results of the optimization for the following system.

A. Integral system with a transport delay

$$G_S(S) = \frac{5}{s(9s + 1)} e^{-2s} \quad (4)$$

In Fig.8 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 is described in [11]. 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.

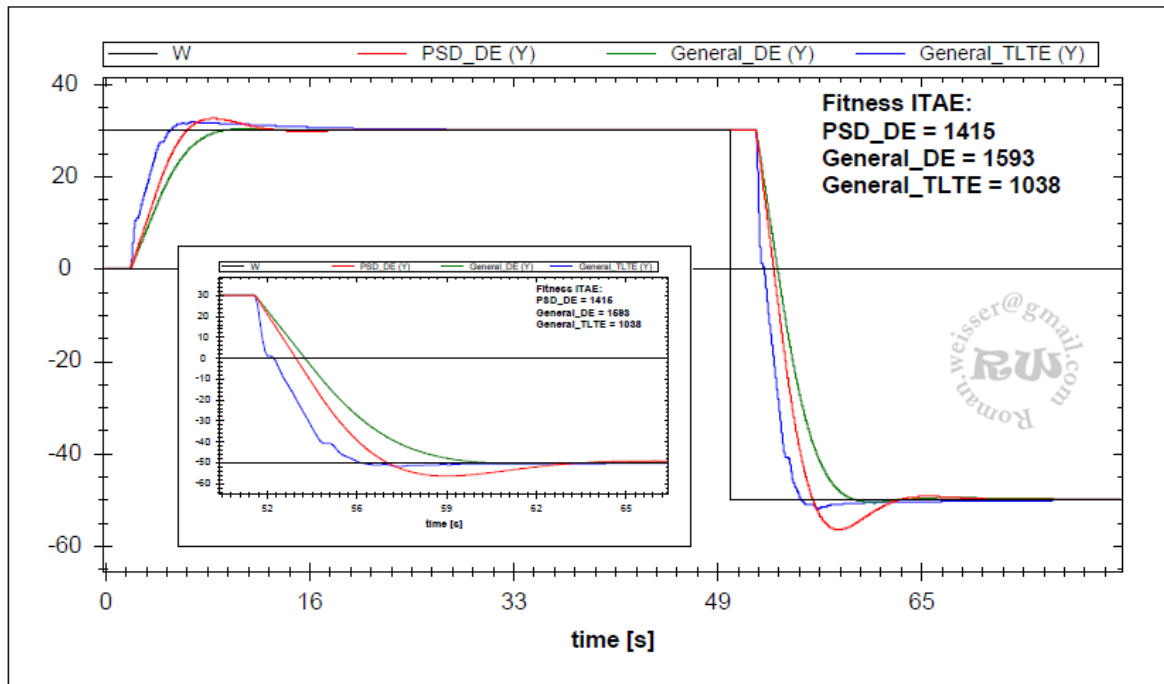


Fig.8. Step response for integration system with time delay

In this case we receive the recurrent control equation with the following form:

$$U_k = (Ek_5 + (((-(Ek_5 * (-4.692)) - (((Yk_5 * 5.148) * Yk_3 * 1.596)) + ((Ek_5 + 22.28) * Ek_2)) * (Uk_2 * (-3.809E-6)))) + ((11.943 * Ek_1) + (((-0.3224) * Uk_3) + ((-18.54) * Ek_2)))) + (Ek_1 + (-0.003999)))$$

VI. 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.

ACKNOWLEDGMENT

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.

REFERENCES

- [1] 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.
- [2] Koza J.R., Genetic Programming: On the Programming of Computers by Means of Natural Selection, The MIT Press, 1992
- [3] 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.
- [4] 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.
- [5] O'Neill, M., Brabazon, A.: Grammatical Differential Evolution, In Proc. International Conference on Artificial Intelligence (ICAI'06) CSEA Press Las Vegas, Nevada.
- [6] O'Neill M. and Ryan C. 2003: Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language Kluwer Academic Publishers.
- [7] Popelka, O.: Parallel Grammatical Evolution for Circuit Optimization. In: Proceedings of MENDEL 2007, Prague, Czech Republic, pp. 88-92 (2007)
- [8] 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.
- [9] Rukovanský I., Evolution of Complex Systems. 8th Joint Conference on Information Sciences. Salt Lake City. Utah. July 21-25, 2005.
- [10] 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
- [11] 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.
- [12] Weisser R., Ošmera P., Two-level Transplant Evolution, In East West Fuzzy Colloquium, 17th Zittau Fuzzy Colloquium 2010