Probability Control Functions Settings in Continual Evolution Algorithm

Zdeněk Buk, Miroslav Šnorek

Dept. of Computer Science and Engineering, Karlovo nám. 13, 121 35 Praha 2, Czech Republic

bukz1@fel.cvut.cz, snorek@fel.cvut.cz

Abstract. *The precise setting of all control parameters of evolutionary algorithms is very important because it affects time needed to find solution, quality of final solution or event the ability to find proper solution, and other technical parameters of computation (e.g. memory requirements), etc. In this paper we are presenting some experiences with settings of probability control functions in continual evolution algorithm (CEA). Evolutionary algorithms are typical examples of nature inspired methods. We will show that the intuitive approach in exact parameters settings, based on our ideas about the nature processes, is not always the best one and we will show the modifications of control functions in CEA algorithm.*

Keywords

continual evolution algorithm, parameters setting

1 Introduction

Continual Evolution Algorithm (CEA) has been theoretically described in our previous work [1, 2, 3]. In [4] more practical experiences are described. The CEA algorithm is under heavy development so some of the initial theoretically expected properties and parameters are modified "on-the-fly" as new experiments are performed.

In first part of the text we will present very brief description of the CEA. Second part of this text is dedicated to description of data structures and control mechanism. Probability and balancing functions in CEA testing will be described. These functions are important part of the CEA, they control whole reproduction/elimination process in the CEA.

2 Theoretical Part

The CEA algorithm is based on standard genetic algorithm (SGA). It extends the SGA by several methods and parameters. It combines genetic operators (representing the evolutionary part of the algorithm) with a gradient optimization method. Evolution in the CEA runs in two relatively independent processes - genetic based process and time-dependent gradient-based process. This two-dimensional evolution is illustrated in figure 1.

The main idea of this approach is to separate the evolution of a structure and behavior (parameters) of individuals. When applied to a neural network construction we can imagine the structure as a topology of network and behavior as a particular weight values setting in such network. When new individual

(e.g. network) is created (born) its structure (topology) is generated and it stays constant in time. Time-dependent gradient based process affects only the behavior part (parameters/weights) of the individual.

Here are some basic properties, principles, and parameters used in CEA:

- variable size of population,
- separated encoding of structure and parameters of models,
- age of each individual,
- sequential replacement of individuals,
- original individuals (parents) are kept in population and optimized together with new individuals (offsprings),
- evolution of individuals in two dimensions – inter-generation evolution and continual adaptation in time dimension (using gradient algorithm),
- probability based control of whole evolutionary process (depending on size of population, quality of individual, and its age).

Note: term *"generation"* here is used only for discrimination of parents and offsprings. It is not the generation as it is defined in SGA.

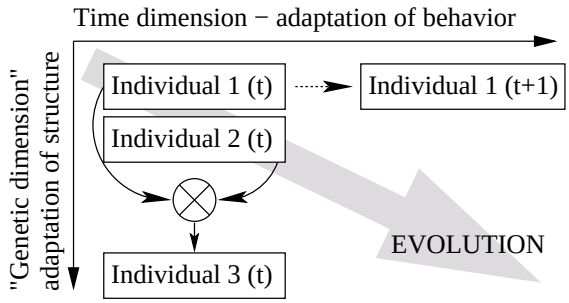


Fig. 1. Two-dimensional evolution in CEA.

2.1 General Structures

This subsection describes the basic data structures used in CEA. Genetic algorithms in general work with some encoding of individuals – each individual represents one solution. In CEA the floating point vector is used for encoding of individuals. This encoding vector is additionally divided into logical parts, representing the structure and behavior of individual – topology and weights setting of neural network represented by the individual. An individual in CEA the individual is represented by the following vector:

$$\bar{x_i} = (a_i, \bar{p_i}, \bar{s_i}, \bar{b_i}), \tag{1}$$

where a_i is the *age* of i-th individual, $\bar{p_i}$ is the initialization *parametric vector* (called instinct), $\bar{s_i}$ is the *structural* parameter and $\bar{b_i}$ is *behavioral vector* of i-th individual, which contains actual set of working parameters of the individual (at the beginning of evolution it is created as a copy of the $\bar{p}$ vector).

The parameters of i-th individual $\bar{x_i}$ are described as follows:

$$\begin{aligned} \bar{p_i} &= (p_{i,1}, p_{i,2}, \ldots, p_{i,u}), \\ \bar{s_i} &= (s_{i,1}, s_{i,2}, \ldots, s_{i,v}), \\ \bar{b_i} &= (b_{i,1}, b_{i,2}, \ldots, b_{i,u}), \end{aligned} \tag{2}$$

where u is the dimension of time (age) dependent parameters vector ($\bar{p}$ and $\bar{b}$) and v is the dimension of the structural parameters vector.

2.2 Probability Functions

The CEA is controlled by several auxiliary parameters that are computed for each individual in population. These parameters are used in the reproduction cycle. The first parameter is the *reproduction probability* which describes the probability, that the i-th individual of age a_i and quality F given by the fitness function, will be used for reproduction operation and that they will produce some new individual (offspring) to the next generation. The reproduction probability is defined as:

$$RP^*(\bar{x_i}) = RP^*(a_i, F(\bar{x_i})), \tag{3}$$

where x_i is the i-th individual that we are computing the probability for, a_i is the age of this individual and function F represents the fitness function – so $F(\bar{x_i})$ represents the fitness value (quality) of the i-th individual.

Parameter *death probability* represents the property that each individual has some maximal age that it can live for. The probability of survival of each individual depends on the quality and actual age of this individual. Here is an example how the death probability is defined:

$$DP^*(\bar{x_i}) = DP^*(a_i, F(\bar{x_i})), \tag{4}$$

where x_i is the i-th individual that we are computing the probability for, a_i is the age of this individual and function F represents the fitness function – so $F(\bar{x_i})$ represents the fitness value (quality) of the i-th individual.

All values signed by $*$ are so called *raw* values. So DP^* is the *raw death probability* and RP^* is the *raw reproduction probability*. The final values DP and RP, which the CEA works with, are computed from the raw values using the *balancing functions*. These functions represent the influence of the size of the population to this size itself – the bigger population will grow slowly (to some limit value, where no new individual will be born) and the smaller population will grow faster.

Final probabilities computation:

$$DP(\bar{x_i}) = BAL_{DP}(N, DP^*(\bar{x_i})), \tag{5}$$

$$RP(\bar{x_i}) = BAL_{RP}(N, RP^*(\bar{x_i})), \tag{6}$$

where N is the size of the actual population and the BAL_{DP} and BAL_{RP} are the general balancing functions.

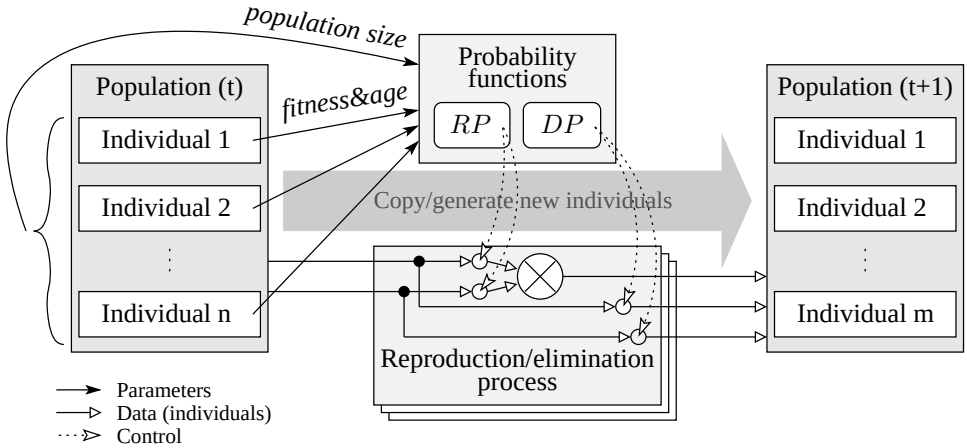


Fig. 2. Diagram of population evolution in the CEA. New generation of individuals is created in process controlled by probability functions. Number (n) of individuals in original population (t) is in general not equal to the number (m) of individual in target (new) population ($t + 1$).

2.3 Evolution Control

Algorithm 1 (The General CE Algorithm)

1. *Initialization*
2. *Repeat until stop condition*
   - *2.1. Evaluate all individuals*
   - *2.2. Reproduction of the individuals with respect to the $RP(\bar{x}_i)$ value*
   - *2.3. Elimination of the individuals with respect to the $DP(\bar{x}_i)$ value*
   - *2.4. Adaptation of the parametrical vector of each individual*
   - *2.5. Update all working parameters and age parameter of individuals*
3. *The result processing*

Figure 2 shows a diagram describing the evolution step, the reproduction/elimination process generating new population of individuals with respect to the probability functions.

3 Experiments

We have chosen the simple experiment "learn to oscillate" to check our implementation and for different probability control functions sets comparison. Using CEA the recurrent neural network was trained to generate defined periodical sequence.

Structure/topology of neural network was generated purely by evolution process in CEA and the particular weights setting was performed by combination of evolution and gradient method.

Following functions sets were tested:

Set 1

$$DP_1^*(a,f) = min(max(a^4 + (1-f)a, 0), 1)/2 \tag{7}$$

$$RP_1^*(a,f) = \frac{e^{-(3a-1.6)^2}}{1+e^{-6(f-0.5)}} \tag{8}$$

$$BAL_{DP,1}(N,DP) = min(max(2N + DP - 1, 0), 1)/10 \tag{9}$$

$$BAL_{RP,1}(N,RP) = min(max(1 - 2N + RP, 0), 1)/5 \tag{10}$$

Set 2

$$DP_2^*(a,f) = min(max(a^4 + (1-f)a, 0), 1)/2 \tag{11}$$

$$RP_2^*(a,f) = min(max(f^2(1-a/2), 0), 1) \tag{12}$$

$$BAL_{DP,2}(N,DP) = min(max(2N + DP - 1, 0), 1)/10 \tag{13}$$

$$BAL_{RP,2}(N,RP) = min(max(1 - 2N + RP, 0), 1)/5 \tag{14}$$

Set 3

$$DP_3^*(a,f) = min(max(a^4 + (1-f)a, 0), 1) \tag{15}$$

$$RP_3^*(a,f) = min(max(f^2(1-a/2), 0), 1) \tag{16}$$

$$BAL_{DP,3}(N,DP) = min(max(2N + DP - 1, 0), 1) \tag{17}$$

$$BAL_{RP,3}(N,RP) = min(max(1 - 2N + RP, 0), 1) \tag{18}$$

Set 4

$$DP_4^*(a,f) = min(max(a^{10} + (1-f)^4 a, 0), 1) \tag{19}$$

$$RP_4^*(a,f) = min(max(f^2(1-a/2), 0), 1) \tag{20}$$

$$BAL_{DP,4}(N, DP) = min(max(2N + DP - 1, 0), 1) \tag{21}$$

$$BAL_{RP,4}(N, RP) = min(max(1 - 2N + RP, 0), 1) \tag{22}$$

Set 5

$$DP_5^*(a,f) = min(max(a^4 + (1-f)a, 0), 1) \tag{23}$$

$$RP_5^*(a,f) = min(max(f^2(1-a/2), 0), 1) \tag{24}$$

$$BAL_{DP,5}(N, DP) = min(max(2N + DP - 1, 0), 1)/10 \tag{25}$$

$$BAL_{RP,5}(N, RP) = min(max(1 - 2N + RP, 0), 1)/5 \tag{26}$$

Set 6

$$DP_6^*(a,f) = min(max(a^4 + (1-f)a, 0), 1) \tag{27}$$

$$RP_6^*(a,f) = min(max(f^2(1-a/2), 0), 1) \tag{28}$$

$$BAL_{DP,6}(N, DP) = \begin{cases} min(max(DP, 0), 1)/2 & \text{for } N > 1/2 \\ min(max(2N \cdot DP, 0), 1)/2 & \text{otherwise} \end{cases} \tag{29}$$

$$BAL_{RP,6}(N, RP) = \begin{cases} min(max(RP, 0), 1)/2 & \text{for } N < 1/2 \\ min(max((2-2N)RP, 0), 1)/2 & \text{otherwise} \end{cases} \tag{30}$$

where a is the age of individual, f is its fitness value, and N represent the actual size of population.

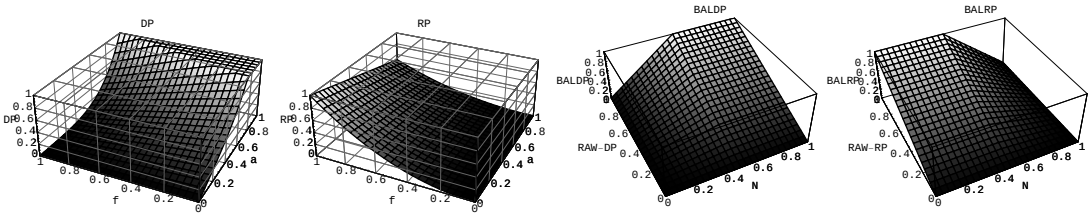


Fig. 3. Visualization of **Set 6** – from left: DP_6^*, RP_6^*, $BAL_{DP,6}$, $BAL_{RP,6}$

In table 1 results of experiments with different probability functions sets are presented. It can be seen that for low number of gradient steps the algorithm gives relatively bad results. It is because in this case the evolution process is used mainly for structure of network building/adaptation. The evolution process is used also for weight setting but it is a minority function. The weights are mainly adapted by gradient algorithm (real-time recurrent learning algorithm was used). In table 1 it can be seen that increasing number of gradient steps gives better results. Different quality of solution for different functions sets can be seen.

Figure 4 shows structure of the neural network we have performed our experiments with CEA on. Figure 4(a) shows the general structure the network – fully connected graph. Figure 4(b) shows an example of trained neural network as it was generated by evolution algorithm. There are 42 connections in the fully recurrent neural network (of this size), using the evolution algorithm number of the connection was reduced to 18, which represents about 57% reduction. In this case (show in figure 4) the number of neurons was not reduced. In our experiments we have used "neutral" fitness function without any explicitly specified requirement to number of neurons reduction.

We have also performed some experiments with pure real-time recurrent learnin algorithm (RTRL) [6, 7] and differential evolution (DE) algorithm [8], selected results are shown in table 3. Because

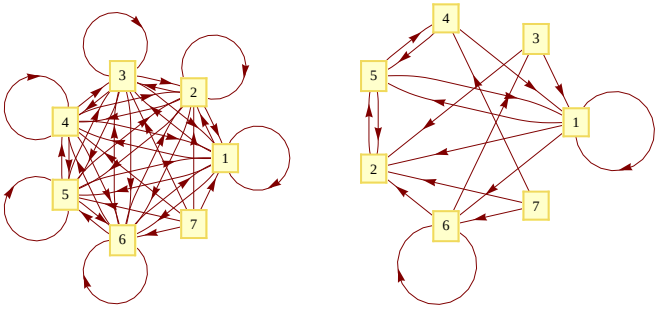
Tab. 1. Results of several probability functions sets testing. Algorithm was tested in 10, 50 and 100 evolution iterations and for different settings of number of gradient steps in each evolution iteration. For each probability functions set we measured number of fitness function evaluations (FC - Fitness Count), population size - PS (average, minimal, maximal) and quality of best solution found (Best).

	10 evolution steps			**50 evolution steps**			**100 evolution steps**		
	FC	**PS**	**Best**	**FC**	**PS**	**Best**	**FC**	**PS**	**Best**
Grad	**Set 1**								
1	191	(17, 10, 26)	0.606683	1416	(28, 10, 34)	0.611728	3036	(30, 10, 35)	0.616973
5	207	(19, 10, 30)	0.615605	1468	(29, 10, 37)	0.779291	3185	(32, 10, 39)	0.950272
10	211	(19, 10, 27)	0.612394	1491	(29, 10, 38)	0.959953	3098	(31, 10, 37)	0.960829
	Set 2								
1	222	(20, 10, 32)	0.608904	1533	(30, 10, 35)	0.609627	3149	(31, 10, 36)	0.617522
5	221	(20, 10, 29)	0.613771	1539	(30, 10, 36)	0.674706	3144	(31, 10, 35)	0.946945
10	265	(24, 10, 34)	0.622972	1553	(30, 10, 37)	0.947883	3209	(32, 10, 38)	0.958767
	Set 3								
1	301	(27, 10, 38)	0.607669	1489	(29, 10, 43)	0.607227	2964	(29, 10, 40)	0.609493
5	306	(28, 10, 37)	0.607749	1487	(29, 10, 40)	0.625760	2959	(29, 10, 43)	0.618908
10	296	(27, 10, 37)	0.609615	1481	(29, 10, 43)	0.668035	2973	(29, 10, 43)	0.907941
	Set 4								
1	304	(28, 10, 38)	0.608464	1474	(29, 9, 42)	0.610587	2954	(29, 10, 40)	0.609087
5	296	(27, 10, 38)	0.610849	1467	(29, 10, 39)	0.618429	2959	(29, 10, 41)	0.619823
10	306	(28, 10, 38)	0.614569	1453	(28, 10, 42)	0.678465	2999	(30, 10, 42)	0.774717
	Set 5								
1	241	(22, 10, 29)	0.609620	1500	(29, 10, 35)	0.612456	3045	(30, 10, 35)	0.614581
5	259	(24, 10, 32)	0.613645	1511	(30, 10, 36)	0.674201	3102	(31, 10, 36)	0.908520
10	249	(23, 10, 30)	0.627898	1516	(30, 10, 37)	0.944204	3265	(32, 10, 38)	0.964902
	Set 5								
1	241	(22, 10, 29)	0.609620	1500	(29, 10, 35)	0.612456	3045	(30, 10, 35)	0.614581
5	259	(24, 10, 32)	0.613645	1511	(30, 10, 36)	0.674201	3102	(31, 10, 36)	0.908520
10	249	(23, 10, 30)	0.627898	1516	(30, 10, 37)	0.944204	3265	(32, 10, 38)	0.964902
	Set 6								
1	249	(25, 10, 42)	0.607035	2063	(41, 10, 50)	0.608738	4327	(43, 10, 51)	0.612297
5	220	(22, 10, 37)	0.611232	2038	(41, 10, 50)	0.673340	4276	(43, 10, 50)	0.884965
10	282	(28, 10, 45)	0.617294	2150	(43, 10, 50)	0.903866	4478	(45, 10, 51)	0.960214

Tab. 2. Results of detailed testing of probability functions set no.6. The table shows number of fitness function evaluations (FS), number of gradient algorithm steps performed (GC), average population size (PS), and quality of best solution found (Best). All these parameters were tested for different maximal population size and number of evolution steps. Number of gradient steps in each evolution iteration was set to 20.

	Maximal population size setting											
Evolution	**10**			**20**			**50**			**100**		
steps	**FC/GC**	**PS**	**Best**	**FC/GC**	**PS**	**Best**	**FC/GC**	**PS**	**Best**	**FC/GC**	**PS**	**Best**
5	57/940	10	0.622	91/1440	18	0.614	245/3900	49	0.639	477/7580	95	0.648
10	106/1900	10	0.687	183/3220	18	0.668	482/8600	48	0.732	958/17120	96	0.682
20	201/3740	10	0.943	365/6780	18	0.867	843/15640	42	0.925	1819/33980	91	0.929
30	301/5700	10	0.950	526/9880	17	0.950	1175/21960	39	0.956	2645/49700	88	0.957
40	397/7500	10	0.952	674/12580	17	0.957	1551/29120	39	0.956	3509/66360	88	0.965
50	503/9580	10	0.961	865/16340	17	0.957	1947/36620	39	0.957	4406/83780	88	0.970

of variable population size and combination of genetic and gradient-based methods in CEA there was highly reduced the number fitness evaluation needed for evolving good-quality individual (network). Differential evolution algorithm uses constant population size, so number of fitness evaluations is also



(a) Fully recurrent neural network (b) Trained recurrent neural network

Fig. 4. An example of recurrent neural network. Left figure shows the fully recurrent neural network with all connections – this is the most general structure that we are building our solution on. Right figure shows the result of evolution process – trained recurrent neural network. About 57% connection has been removed by evolution process. Neuron number 7 is "fake" neuron representing the 1 value for threshold realization. Output of the network is taken from the neuron number 1.

constant for particular number of individuals and number of generation setting. Gradient-based algorithm (RTRL) works with only one network and it needed less steps than the CEA. In CEA the average size of population was 24 although the maximal size was the same as in DE and it was set to 100.

The RTRL algorithm is not able to create optimal topology of network. It always uses the fully connected network. DE also optimized the whole weight matrix representing the fully recurrent neural network. CEA reduced some connection, see figure 4(b).

Tab. 3. Selected experiments results. Comparison of number of iterations needed in particular algorithms to train the network with comparable result (errors). DE-differential evolution, CEA-continual evolution algorithm, RTRL-real time recurrent learning. The table shows numbers for whole population and for the best individual for comparison. Shown values are approximate – they come from several runs of algorithms.

	DE	CEA	RTRL
Total for whole algorithm run.			
No. of fitness evaluations	25 000	480	×
No. of gradient steps	×	9 000	500
For best solution (one individual).			
No. of fitness evaluations	500	17	×
No. of gradient steps	×	340	500

4 Conclusion and future work

A short description of CEA algorithm was presented in this paper. Several sets of probability control functions was shown and tested. As a result of our experiments we could say that the reproduction probability plays less significant role that we expected, while the death probability seems to be more important. This relation probably depends mainly on the particular crossover operators used. We used simple crossover operator combined with mutation. Particular settings of probability functions and

other parameters in CEA algorithm is difficult task and it is still the theme of the future work. One of the approaches could be usage of genetic algorithm or other evolutionary method for searching of the remaining parameters of this algorithm.

Acknowledgments

This work is supported by the internal grant of Czech Technical University in Prague, IG: CTU0706813 *Development of the "continual evolution" optimization algorithm and its application to the artificial neural networks adaptation*.

References

[1] Buk, Z., Šnorek, M.: A brief introduction to the continual evolution algorithm. In *Proceedings of 6^{th} International Conference APLIMAT 2007, Part III*, pages 71–76, Bratislava, February 2007, ISBN: 978-80-969562-6-5.

[2] Buk, Z.: Survey of the computational intelligence methods for data mining. *Technical Report DCSE-DTP-2007-01, Czech Technical University in Prague, FEE*, CTU Prague, Czech Republic, 2007.

[3] Buk, Z., Šnorek, M.: Nature Inspired Methods for Models Creation and Optimization In *Proceedings of 41th Spring International Conference MOSIS'07*, pages 56–62, Ostrava: MARQ, 2007, ISBN: 978-80-86840-30-7.

[4] Buk, Z., Šnorek, M.: Continual Evolution Algorithm for Building of ANN-based Models. To appear in *proceedings of the 6^{th} EUROSIM Congress on Modelling and Simulation*, Ljubljana, Slovenia, 2007.

[5] X. Yao. Evolving artificial neural networks, 1999.

[6] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989.

[7] R. J. Williams. Gradient-based learning algorithm for recurrent networks and their computational complexity. *Y. Chauvin and D.E. Rumelhart (Eds.) Back-propagation: Theory, Architectures and Applications.*, 1995.

[8] Lampinen Jouni A., Price Kenneth V., Storn Rainer M. *Differential Evolution A Practical Approach to Global Optimization*. 2005. ISBN 978-3-540-20950-8.