Juraj SPALEK[*] Michal GREGOR[**]

ADAPTIVE SWITCHING OF MUTATION RATE FOR GENETIC ALGORITHMS AND GENETIC PROGRAMMING

Abstract

The paper concerns the application of Genetic Algorithms and Genetic Programming to complex tasks such as automated design of control systems, where the space of solutions is non-trivial and may contain discontinuities. An adaptive value-switching mechanism for mutation rate control is proposed. It is shown that the proposed mechanism is useful in preventing the search from getting trapped in local extremes of the fitness landscape.

INTRODUCTION

Genetic Algorithms represent a well-known optimization method recognized in particular for its flexibility in representation of solutions and for its ability to produce reasonably fit results in a reasonable amount of time. Genetic Programming applies the theory of Genetic Algorithms to evolving computer programs, usually represented by syntactic trees.

There is a multitude of research papers that aim to improve convergence and robustness of both methods. Some of these concentrate on parameter control, that is to say on setting and modifying various parameters of the algorithm.

This paper presents an adaptive value-switching mechanism for control of the mutation rate, which aims to decrease the probability that the search will become trapped in local maxima by increasing mutation probability to a high value once such scenario is detected.

GENETIC ALGORITHMS AND GENETIC PROGRAMMING

Although the methods in question are relatively well known, let us first present some fundamental information about both – Genetic Algorithms (GA) and Genetic Programming (GP).

[*] Prof. Ing. Juraj Spalek, PhD. – Department of Control and Information Systems, Faculty of Electrical Engineering, University of Žilina, Univerzitná 1, 010 26 Žilina, Slovak Republic, juraj.spalek@fel.uniza.sk

[**] Bc. Michal Gregor – Department of Control and Information Systems, Faculty of Electrical Engineering, University of Žilina, Univerzitná 1, 010 26 Žilina, Slovak Republic, o.m.gregor@gmail.com

Genetic algorithms represent one of the several computational techniques based on simulation of evolution, a process based on the principle of *natural selection*, that is, on the *survival of the fittest*. The genetic algorithm operates on a population of individuals.

The individuals represent various solutions of a specific problem. The main principle of the algorithm is as shown in figure 1.

The first step is to generate the initial population – this typically involves generating a group of random individuals. The next step is to perform evaluation of those individuals, which enables the algorithm to compare the individuals to each other and, furthermore, to introduce the survival of the fittest: the individuals with the best scores (also known as *fitness* in the genetic algorithm terminology) are the most likely* to participate in *reproduction*, that is, in forming the next generation. This is analogous to the natural selection process, in which the fitter individuals have greater chance to survive and to reproduce.

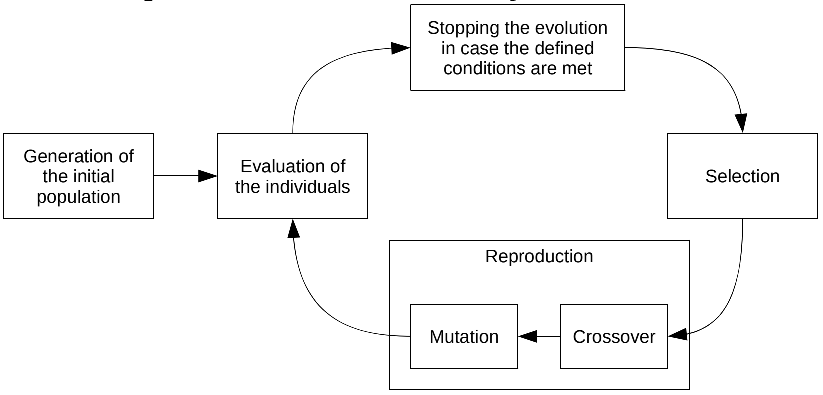
Stopping the evolution in case the defined conditions are met
Generation of the initial population
Evaluation of the individuals
Selection
Reproduction
Mutation
Crossover

Fig. 1. The general principle of genetic algorithms

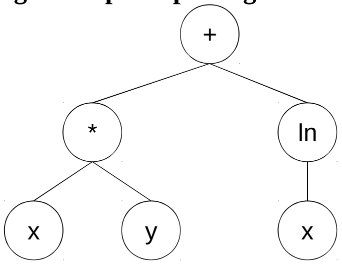

Fig. 1. A simple example of a syntactic

Genetic programming (GP) is a technique developed by John Koza (see *Genetic Programming: On the Programming of Computers by Means of Natural Selection* [1]). It applies the theory of Genetic Programming to the task of evolving computer programs. The main idea of Genetic Programming is the way in which the individuals are represented – by syntactic trees (also known as parse trees). Fig. 2 shows a simple example of a syntactic tree that codes the expression $x.y + \ln x$.

* However, we usually refrain from directly choosing the best *n* individuals as that tends to reduce diversity, which leads to getting trapped in a local extreme.

For syntactic trees crossover is usually done by swapping 2 randomly selected sub-trees of the 2 parent individuals, while mutation may be implemented by replacing a randomly selected sub-tree by a newly generated one. For a more detailed introduction to the problem refer to [1] or [2].

THE ARTIFICIAL ANT PROBLEM

The artificial ant problem described by John Koza in [1] is essentially a trail-following task. The actor – an artificial ant – is supposed to navigate in an environment following an irregular path consisting of pieces of food which it collects. The ant has very limited sensing capabilities – it only sees a single tile right in front of it. John Koza successfully solves the problem by applying Genetic Programming†.

In our work we have set some additional requirements concerning the form of the solution – the evolved controller should, when executed, return the action that the ant is to execute next instead of calling functors that directly execute the action and wait for its completion. The set of terminals contains persistent variables and the controller has access to a pre-set number of its previous inputs and outputs.

Controllers based on such mode of execution seem to be much more difficult to evolve than those originally proposed by Koza. The search usually gets trapped in a local maximum from which it is often unable escape.

EXISTING APPROACHES TO PARAMETER CONTROL

In some applications based on the theory of genetic algorithms, the optimization task may be so difficult – with a complex space including a great number of local optima in which the search process can get trapped – that additional techniques may be required to find the global optimum. Genetic programming does in a multitude of tasks serve as an especially good example of the problem, as it evolves computer programs and it is obvious that two very similar computer programs may produce drastically different results and thus the space of solutions is highly complex.

Among the approaches that aim to prevent getting trapped in a local optimum are adaptive schemes that observe various parameters of the algorithm or the search process itself and using the observed values adapt some of the parameters. The approaches to parameter setting can basically be divided into the following categories [3], [4]:

- static parameter control,
- dynamic parameter control,
- adaptive parameter control,
- self-adaptive parameter control.

Static Parameter Control

The common feature of approaches falling into this category is that the setting they provide remains constant for the entire duration of the evolutionary process. There are many works analysing the problem of finding optimum settings for parameters like mutation probability and

† See [1] for detailed information about the solution.

crossover probability. Some of these are listed in [3], e.g. the work of Mühlenbein, which proposes the following formula for the mutation probability:

$$p_m = 1/L\,, \tag{1}$$

where L is the length of the bit string by which the individual is represented.

Dynamic Parameter Control

As stated in [4] dynamic parameter approaches typically prescribe a deterministically decreasing schedule over a number of generations and provides a formula for mutation probability derived by Fogarty:

$$p_m(t) = \frac{1}{240} + \frac{0.11375}{2^t}\,, \tag{2}$$

where t is the generation counter.

Articles [3], [4] both refer to a more general expression derived by Hesser and Männer:

$$p_m(t) = \sqrt{\frac{\alpha}{\beta}} \times \frac{\exp\left(\frac{-\lambda t}{2}\right)}{\lambda\sqrt{L}}\,, \tag{3}$$

where α, β, γ are constants, λ is the population size and t is the generation counter and L is again the length of the bit string.

Adaptive Parameter Control

Adaptive parameter control techniques monitor the search process itself and provide feedback. Some examples can be found in [5]. The authors propose the following formulas for crossover and mutation probability respectively:

$$p_c = \begin{cases} k_1 \dfrac{f_{max} - f'}{f_{max} - \bar{f}} & f' > \bar{f} \\ \\ k_3 & f' \le \bar{f} \end{cases} \tag{4}$$

$$p_m = \begin{cases} k_2 \dfrac{f_{max} - f}{f_{max} - \bar{f}} & f > \bar{f} \\ \\ k_4 & f \le \bar{f} \end{cases} \qquad (5)$$

where f is the fitness value of the individual to be mutated, f' is the larger of the fitness values of the individuals to be crossed and k_3 and k_4 are constants. It is required that k_1 and k_2 be less than 1.0 in order to constrain p_c and p_m to the range of $\langle 0,1 \rangle$. The $p_c = k_3 \quad f' \le \bar{f}$ and $p_m = k_4 \quad f \le \bar{f}$ expressions are to prevent crossover and mutation probabilities from exceeding 1.0 for suboptimal solutions.

Authors of [5] also observe that p_c and p_m are zero for the solution with maximum fitness and that $p_c = k_1$ for $f' = \bar{f}$, while $p_m = k_2$ for $f = \bar{f}$. For further details and for information concerning setting the values of the constants refer to [5].

Self-adaptive Parameter Control

When using the self-adaptive parameter control approach, parameters such as mutation rate and crossover probability of each individual are part of its genome and are evolved with it. As stated in [4], the idea behind this is that a good parameter value will provide an evolutionary advantage to the individual. For further reference see [3] or [4].

ADAPTIVE VALUE-SWITCHING OF MUTATION RATE

Motivation

Most of the existing parameter setting mechanisms, as presented in the previous chapter, either focus on setting GA-specific parameters such as length of the bit string (e.g. rule (1)), or are not adaptive (e.g. (2) and (3)). The adaptive mechanism described in [5] (formulas (4) and (5)) seems more fit to the task because it implements certain form of convergence detection based on comparison of the maximum and average fitness values. However this approach does little to solve the problem of getting trapped in a local optimum as the method does not discern between local and global optima.

Furthermore – as mentioned hereinbefore – equations (2) and (3) assign the best individual zero crossover and mutation probabilities, while assigning high probabilities to less fit individuals. The reasoning behind this is that the less fit individuals can safely be disrupted by high mutation rates and recombined by crossover (thus employing the solutions with subaverage fitness to search the space [5]), while the highly fit individuals should be preserved.

However, such approach has a very obvious downside which the authors do not seem to address – the highly fit individuals obviously contain the most excellent genetic material available and by disallowing mutation and crossover for these individuals the genetic code they carry becomes isolated and is not used to generate new solutions.

Description of the Proposed Adaptive Mechanism

The idea that the most fit solutions should survive crossover and mutation unmodified is valid, yet that feature can be enforced by using elitism‡. Keeping that in mind we propose a different adaptation scheme in order to address the other issues. The main idea is that the mutation probability should be increased to a high value when the search has become trapped in an extreme so as to provide the search process with new genetic material some of which may previously have been unavailable. To determine whether the search has become trapped the adaptive mechanism observes the change of average fitness in time.

To describe the solution in more detail – the algorithm works with 2 values of mutation probability – the normal value and the high value. The algorithm switches from the normal value to the high value once the trigger criterion activates.

The trigger criterion itself is based on a measure that we will herein term a *delta sum*:

$$\Delta S_i = \alpha . \Delta S_{i-1} + \frac{\bar{f}_i - \bar{f}_{i-1}}{\bar{f}_i}, \tag{6}$$

where ΔS_i is the delta sum in generation i and $\bar{f}_i$ is the average fitness in generation i and α is the feedback coefficient (the experiments have been carried out for $\alpha = 0.4$).

If the delta sum is lower than a pre-set value for a predefined number of generations, that is to say the increase of average fitness in the last few generations is low, indicating that the search has become trapped§ – the mutation probability is set to its high value so as to provide the search with new genetic material. As mentioned before, when used in conjunction with elitism it is guaranteed that the best solution is not destroyed by the high mutation probability.

The mutation probability is reset back to its normal value when at least one of the following conditions is true:

- the average fitness increases enough to produce a sufficiently large delta sum;
- the maximum fitness increases;
- mutation has been set to its high value for at least n generations.

The *n*-generation limit is to ensure that the activation does not go on indefinitely (with the high mutation probability it is not very likely that the average fitness will increase enough to satisfy the first condition and maximum fitness may not increase as well).

It has been observed that average fitness typically decreases when the criterion activates because the search process is to a large extent disrupted by the high mutation probability. However after the *n*-generation limit forces the mutation rate back to its normal value, average fitness tends to increase rapidly, thus usually moving away from the local extreme.

Experimental Results

Several experiments have been carried out – Fig. 3 shows performance of the search algorithm with the AGA adaptive mechanism proposed in [5] with constants set according to

‡ The best individual is copied to the next generation unmodified.

§ This may also indicate convergence to the global maximum, it is, however, hardly possible to tell global and local maxima apart unless the algorithm is provided with additional problem-specific data.

recommendations. It also shows performance of the search algorithm without any adaptive mechanism and with the adaptive mechanism proposed in this paper. The maximum fitness value achieved is shown for each of the 5 runs displayed.

As shown, search achieves suboptimal results when running with no adaptive mechanism. This can be ascribed to its inability to escape from local extremes. With no adaptive mechanism the search has not found the global optimum (fitness = 89) in any of the 5 runs.

As expected, the AGA mechanism has caused further deterioration and its results are even worse than those produced in the previous case.

The Value-switching adaptive mechanism proposed in this work improves the process of search – in 2 of the runs the global optimum is found, yet in certain cases not even the high mutation rate is guaranteed to help the search escape from the local maximum (runs 2, 3, 4).

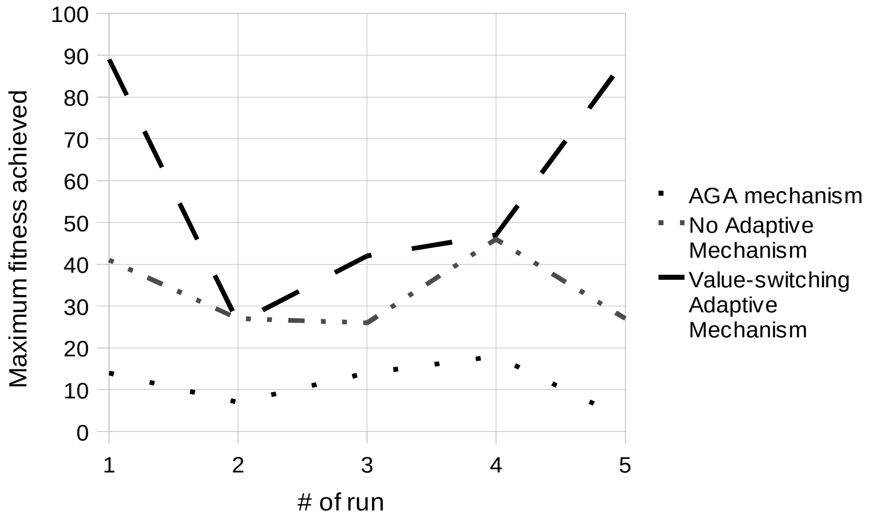

Fig. 2. Comparison of the AGA Adaptive Mechanism and the Value-switching Adaptive Mechanism

Suggestions for Further Work

It has been shown that the adaptive mechanism described in this work is able to effect considerable improvements and that it is able to some extent prevent getting trapped in local maxima. Further experiments should now be carried out in order to ascertain that the principle is valid for a wider range of tasks.

It has also become apparent that even with the high mutation rates it is not always guaranteed that the search will indeed escape from the local maximum. Value-switching, or piecewise continuous relationships for other parameters could perhaps help to alleviate the problem – this issue requires further investigation.

CONCLUSION

It is well known that search processes based on genetic algorithms and genetic programming are prone to getting trapped in local maxima when exploring highly complex spaces.

As shown in the paper, search process based on the standard genetic programming approach fails to find the global optimum when applied to the modified version of the artificial ant problem.

This paper investigates the problem and proposes an adaptive mechanism for mutation rate control, which should help the search to escape from local extremes. As shown, the results are considerably better than those of the standard genetic programming approach.

Although the results are significantly better, even the adaptive value-switching of mutation rate as here proposed cannot always guarantee that the process will escape from a local maximum. It is possible that value-switching, or piecewise continuous relationships for other parameters could help to alleviate the problem. Such approaches could provide area for further research.

This paper is a part of a more comprehensive research supported by: ASFEU 26220220049.

REFERENCES

1. KOZA J. R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press. Cambridge, Massachusetts, 1998. ISBN 0-262-11170-5
2. HYNEK J.: *Genetické algoritmy a genetické programování*. Grada Publishing, a. s. Praha, 2008. ISBN 978-80-7300-218-3
3. EIBEN Á. E., ROBERT, H., MICHALEWICZ, Z.: *Parameter Control in Evolutionary Algorithms*. IEEE Transactions of Evolutionary Computation: 3, 1999. http://www.gpa.etsmtl.ca/cours/sys843/pdf/Eiben1999.pdf
4. THIERENS, D.: *Adaptive mutation rate control schemes in genetic algorithms*. Proceedings of the 2002 IEEE World Congress on Computational Intelligence: Congress on Evolutionary Computation, 2002. http://dynamics.org/~altenber/UH_ICS/EC_REFS/GP_REFS/CEC/2002/GP_WCCI_2002/7315.PDF
5. SRINIVAS, M., PATNAIK, L. M.: *Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms*. IEEE Transactions on Systems, Man and Cybernetics: 24, 1994. http://eprints.iisc.ernet.in/archive/00006971/02/adaptive.pdf