Computational time for 100 runs of algorithm in seconds is on the vertical axis, the length of the original words is on the horizontal axis.

Conclusion. To sum it up, from the figures above it could be seen that the ASAWT with and without relevance function works faster than the Wagner-Fisher algorithm. The final lists of similar words are almost identical. The two-step system allows to keep the accuracy at the same level.

However, the algorithm has some drawbacks, namely there are no weights modeling the closeness of the pronounced sounds. This problem is supposed to be solved in a future work.

Acknowledgements. This work is partly supported by the DAAD (German Academic Exchange Service).

References

1. Zablotskiy S., Shvets A., Sidorov M., Semenkin E. and Minker W. Speech and Language Resources for LVCSR of Russian. International Conference on Language Resources and Evaluation (LREC), Istanbul, Turkey, 2012. May.

2. Smetanin N. (2011, March 24). Fuzzy string search. *Nikita's blog. Search algorithms, software development and so on.* Retrieved July 30, 2013, from http://ntz-develop.blogspot.ru/.

3. Levenshtein V. I. 1966. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10, February.

4. Wagner R. A., Fischer M. J. The string-to-string correction problem. J. ACM. 1974, vol. 21, № 1, p. 168–173.

5. Wikipedia. Similarity index (coefficient). *Wikipedia. The free encyclopedia.* Retrieved July 30, 2013, from http://ru.wikipedia.org/ (in Russian).

6. Karakhtanov D. S. Using of fuzzy search algorithm in processing of data for credit institutions. Audit and financial analysis. 2010, vol. 2 (in Russian).

УДК 519.6

SELF-ADJUSTED EVOLUTIONARY ALGORITHMS BASED APPROACH FOR AUTOMATED DESIGN OF FUZZY LOGIC SYSTEMS

V. V. Stanovov, E. S. Semenkin

Siberian State Aerospace University named after academician M. F. Reshetnev
31, Krasnoyarsky Rabochy Av., Krasnoyarsk, 660014, Russian Federation
E-mail: vladimirstanovov@yandex.ru, eugenesemenkin@yandex.ru

A new approach to form a fuzzy logic system with evolutionary algorithms is introduced. Several algorithms were implemented as programs, efficiency measurements were made with and without self-adjustment. A new data representation method was developed for fuzzy rules base coding in genetic programming method by using reverse polish notation principle. The genetic algorithm was adapted to adjust linguistic variables semantics. The efficiency of developed algorithm was shown and compared to analogies on several test regression problems and real classification problems.

Keywords: genetic algorithms, genetic programming, self-adjustment, fuzzy logic system design.

САМОНАСТРАИВАЮЩИЙСЯ ЭВОЛЮЦИОННЫЙ АЛГОРИТМ ДЛЯ АВТОМАТИЗИРОВАННОГО ФОРМИРОВАНИЯ СИСТЕМ НА НЕЧЕТКОЙ ЛОГИКЕ

В. В. Становов, Е. С. Семенкин

Сибирский государственный аэрокосмический университет имени академика М. Ф. Решетнева
Российская Федерация, 660014, Красноярск, просп. им. газ. «Красноярский рабочий», 31
E-mail: nihilanht@mail.ru, eugenesemenkin@yandex.ru

Представлен новый подход для формирования систем на нечеткой логике эволюционными алгоритмами. Несколько алгоритмов были реализованы в качестве программ, проведены измерения эффективности с самонастройкой и без таковой. Был разработан новый метод представления данных для кодирования нечетких правил в генетическом программировании на основе принципа обратной польской записи. Генетический алгоритм был адаптирован для настройки семантики лингвистических переменных. Эффективность разработанного алгоритма была показана в сравнении с аналогами на некоторых тестовых регрессионных задачах и практических задачах классификации.

Ключевые слова: генетический алгоритм, генетическое программирование, самонастройка, проектирование систем на нечеткой логике.

Fuzzy logic systems are a type of intellectual information technologies and for today they are used widely a variety of problems. Among them, there are car speed control systems, handwritten text recognition systems, robots control systems and different data analysis problems. But using them even more widely is difficult because their forming process takes a lot of time and human resources. So, developing algorithms that would allow us to form such systems automatically will help to increase the efficiency of using and the availability of such systems.

Fuzzy logic systems are also very useful as they represent knowledge as rules, which are commonly easy to understand for a human. And that is why most of the times they are formed by experts, i.e., using their knowledge. But knowledge extraction procedure may not get hidden expert knowledge, and this is one of the disadvantages of such an approach.

Automatic fuzzy logic system forming is formulated as an optimization problem, i.e., choosing some optimal rules base in terms of an efficiency criterion. Solutions representation in computer memory is also a difficult algorithmic and computational problem.

The genetic programming is a method for these problems solving. It has the flexible data representation using graphs, and it allows choosing the best solution automatically. Still, it has several disadvantages, for example, the necessity for optimal setting of genetic operators. The algorithm's efficiency depends on the set of genetic operators used, and for every research problem this combination of genetic operators may be different. To choose the best operators, several self-adjustment methods can be used.

But forming a fuzzy logic rules base is not enough to form a fuzzy system. Defining linguistic variables and their semantics is also necessary. The linguistic variables are used in the fuzzy logic systems to determine the fuzzy values (terms) like "weak", "medium" and "strong". Basically, the number of terms and their position is determined by experts, but this definition may be not ideal. To adjust the position of terms, a self- adjusted genetic algorithm can be used.

The first part of the paper describes the genetic algorithm, the second deals with genetic programming, and the third describes the fuzzy logic systems generating algorithm.

Self-adjusted genetic algorithm. To test the efficiency of self-adjustment methods, a standard genetic algorithm (GA) was developed and implemented separately. The operators of this algorithm are presented in table 1.

Let's consider two self-adjustment methods [1]: Population-Level Dynamic Probabilities (PDP) and Individual-Level Dynamic Probabilities (IDP). These methods were developed to adjust mutation type in genetic programming algorithm.

The main idea of PDP method is that operator's probabilities depend on success of their application. So, after every genetic operator application the fitness of the obtained offspring is compared to parents' fitness. If the fitness increased, the operators applied receive a reward, i.e., their application probabilities increase. New probabilities depend on the success rates, number of operators used and are calculated as following:

$$p_i = p_{all} + \left[\frac{r_i \cdot (100 - n \cdot p_{all})}{scale} \right],$$

$$r_i = \frac{success_i^2}{used_i},\ p_{all} = \frac{20}{n},\ scale = \sum\nolimits_{j=1}^{n} r_j .$$

Table 1

Groups of genetic operators for GA

Operator groups	Selection	Crossover	Mutation
Operators	Proportional Rank Tournament	Single-point Two-point Uniform	Weak Average Intensive

Here *success$_i$* is the number of successful *i-th* operator applications; *used$_i$* is the number of operators' applications. Also, there can be different success rates. For example, if the offspring is better than both parents, then it is clearly successful (*success = success+1*), and if it is better than only one parent, then it is partly successful (*success = success+0,5*); if it is worse than both parents, then it is unsuccessful (*success* values does not change).

The IDP method was described so that it can only be applied to choose the mutation type. It was modified to choose the crossover type. The essence of the method is that every individual in the population has several counters cnt_j^i that show the number of unsuccessful applications of operator *i* for individual *j*. These values are used to calculate the probabilities:

$$p_i = p_{all} + \left[\frac{(\max_{1\le k\le n} cnt_j^k + 1 - cnt_j^i) \cdot (100 - n \cdot p_{all})}{n \cdot (\max_{1\le k\le n} cnt_j^k + 1) - \sum\nolimits_{k=1}^{n} cnt_j^k} \right],$$

$p_{all} = \dfrac{20}{n}$, *n* is the number of operators, *m* is the operator number.

As this method requires one or two parent individuals, it cannot be used to choose the selection type. I.e., the probabilities needed to choose the operator depend on the individuals we use, and so to be able to apply IDP scheme, there should be at least one individual. Before the selection process, we do not know which *cnt* values to take. Taking *cnt* values from a random individual from the population is not good, because of the idea that every individual has its own *cnt* values.

So, when the IDP method is used to choose the crossover and mutation types, the PDP method is used to choose the selection type. Moreover, the same as for PDP method, there should be a way to differ the operators' success. So, if the offspring is worse than parents then $cnt_j^i = cnt_j^i + 1$; if it is worse than one of them then cnt_j^i does not change, and if it is better than both parents then $cnt_j^i = 0$.

After the calculation, the p_i values are mapped into interval (0,1). p_{all} value is the minimum value and to is needed to make the worst operator using probability not become zero, because in this case the operator will not be used at all.

The self-adjustment efficiency was measured in comparison to conventional genetic algorithm with all parameter combinations (45 combinations as there were five selection types used, including three tournament selections with tournament sizes equal 2, 5 and 9). The test functions used are complex for standard optimization methods. They have lots of local optima, plateaus, and global optima within them.

The reliability value was used to determine the efficiency of all algorithms; it was measured as the ratio of number of cases when the algorithm found the goal with certain accuracy to the number of algorithm runs (100 runs for every parameter combination). The results of all methods performance on one test function is presented on figure 1. For all other test functions these results look approximately so as well. The aver. GA reliability is calculated as an arithmetical mean of all reliabilities measured at every parameters combination.As can be seen from the graph, both self-adjustment methods show nearly the same reliability, and they are much better than the average genetic algorithm. Still, the best combination of parameters for the selected test function gives higher reliability value than self-adjusted GA. As a result, we may say that the presented self-adjustment methods are effective and can be used to choose genetic operators during the algorithm execution on the problem in hand.

Self-adjusted genetic programming. Genetic programming (GP) [2] has the same problems with operators used. So, the same self-adjustment methods can be applied to it. To see if these methods are efficient with genetic programming, the algorithm was implemented separately to solve symbolic regression problems.

The symbolic regression problem is finding an optimal mathematical equation, which can be used to approximate some numerical dependence between several variables. These equations can be used to solve regression and classification problems. They consist of the following operations (functional set): *+, -, *, /, sin, cos, tan, atan, ln, sqrt, pow* and include input variables and constants as terminal set. An example of symbolic equation is:

$$(5*X)*\sin(Y)+(Y+1)*\cos(X).$$

Coding such equations in computing memory requires building trees, as *+, -, *, /, pow* are binary operations. To ease this coding, a reverse polish notation principle was used, in which the operands stand before the operation sign. Such data representation allows to code equations into strings without brackets. An example of symbolic equation in reverse polish notation is:

$$5X*Y\sin*Y1+X\cos*+.$$

Also such an implementation allows accessing any part of the equation without going throw the entire tree and avoids often problems with the computer memory allocation and cleanup. The only problem with reverse polish notation is that subtrees extraction procedures are more complex and take more time during crossover and mutation.

Two self-adjustment methods – PDP and IDP can be applied in this case without changes.

Table 2

Groups of genetic operators for GP

Operator groups	Selection	Crossover	Mutation
Operators	Proportional Rank Tournament	Standard Single-point	Point Growing

The fitness function for GP also includes penalties that depend on the equation length and number of variables used in it. The algorithm was tested to approximate the same test functions, used for genetic algorithm testing. So, finding the exact solution is hardly possible. In the figure below the error values on each of these functions are compared for different GP algorithms.

Same as for the genetic algorithm, the self-adjustment procedure allows overcoming the efficiency of average algorithm, and is a little worse, than the algorithm with the best combination of parameters.

Genetic programming for fuzzy logic systems forming. The genetic programming algorithm for forming fuzzy logic systems uses the Pittsburg approach for forming rule bases, i.e., it forms complete rule bases, not separate rules. The rule bases are presented as trees, as shown at figure 3. The selection operator does not change in this case, while the crossover operator cannot use the root as a point for crossover unlike standard GP.

The root of the tree means logical operation *"or"*, and it combines different rules. Every subtree of the root is a separate rule. The nodes within every subtree of the root can be from terminal or functional set. If the node is from the functional set, it means the logical operation *"and"*, combining parts of the rule. If the node is from the terminal set, it is a part of the rule, and says *"if variable x equals y"*. A combination of such parts forms a rule. The fuzzy inference value is kept in the functional or terminal node, connected to the root.

So this tree codes the following rules (read from left to right):

If x1=1 and x2=1 then output=1;
If x3=2 then output=3;
If x1=2 and x2=3 and x3=3 then output=2;
If x3=1 then output=3;
If x2=2 and x1=3 then output=1.

This coding allows forming compact rule bases with needed number of rules and low memory requirements. The same as for standard GP, here selection, crossover and mutation operations can be applied. The only one difference is that the root of the tree cannot be the crossover point.

The second part of forming a fuzzy logic system is adjusting the linguistic variables semantics. This includes determining terms position, but not the number of terms.

The algorithm was tested on several test functions, the same as genetic programming for solving symbolic regression problem. The comparison of errors obtained on every function is shown at fig. 4. The self adjustment procedure used for fuzzy GP is PDP.

As one may see, the algorithms have different behavior on different functions, but their overall efficiency is close enough.

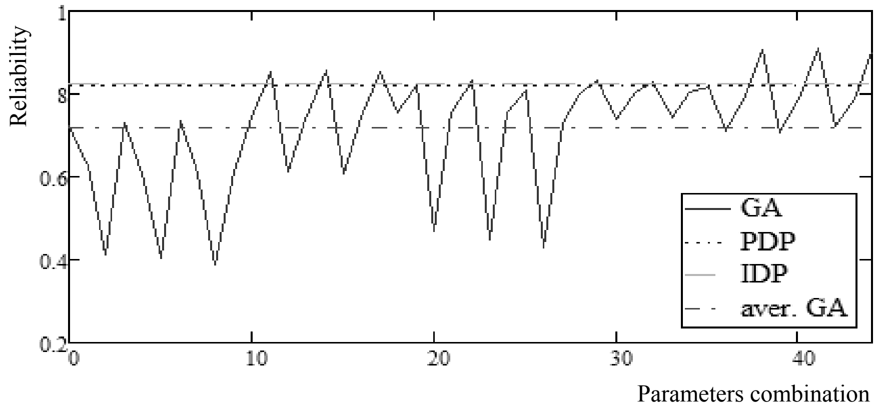

Fig. 1. Standard GA reliability compared to self-adjusted GA reliability

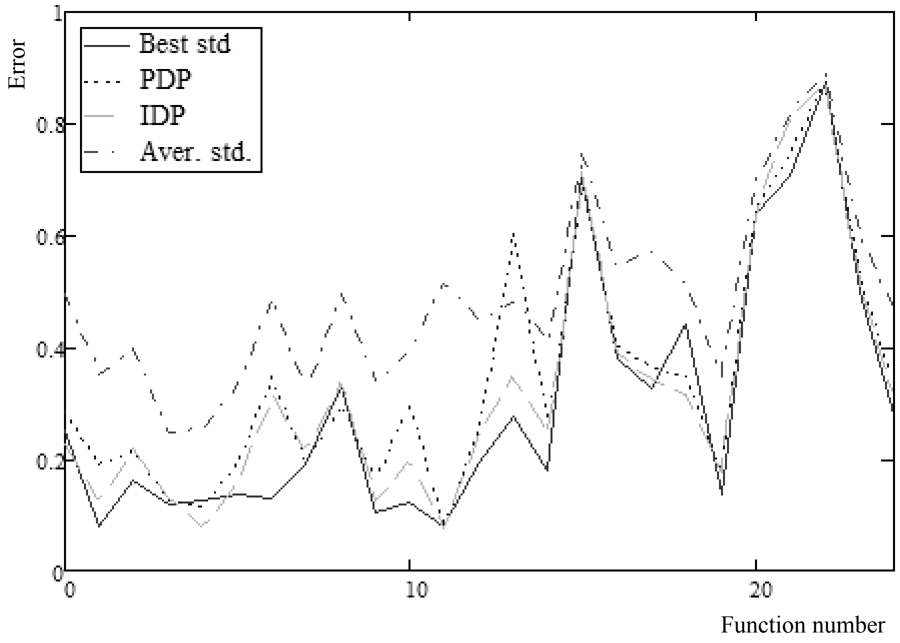

Fig. 2. Standard GP error compared to self-adjusted GP errors

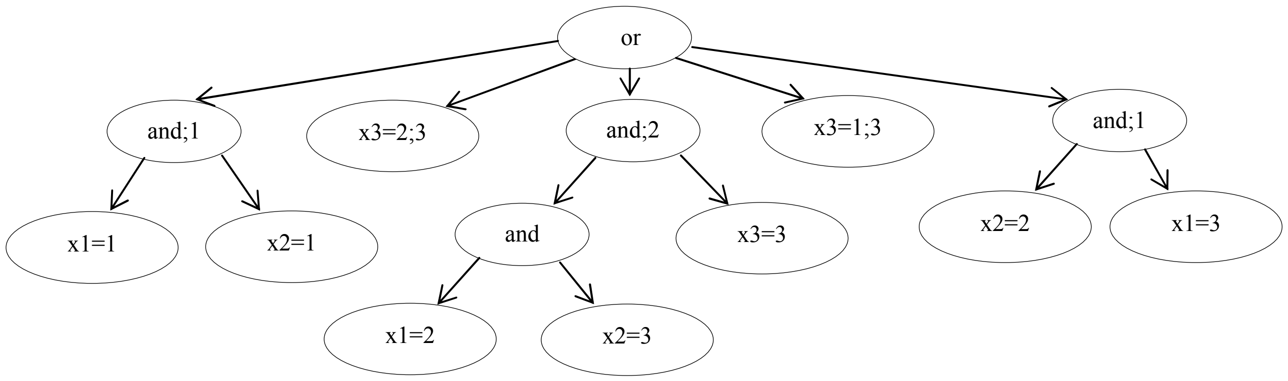

Fig. 3. Rules base representation

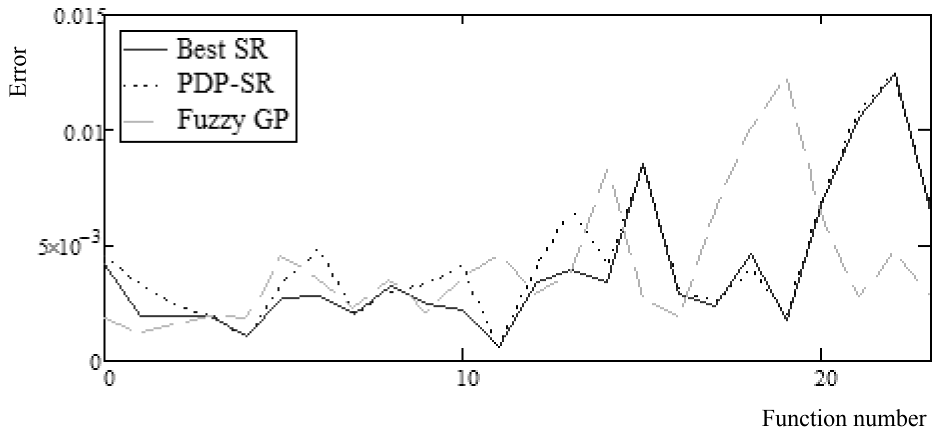

Fig. 4. Comparison of symbolic regression (SR) GP and fuzzy logic forming GP

Table 3

Comparison of different methods [3] solving classification problems

Algorithm name	Australian credit	German credit	Algorithm name	Australian credit	German credit
SCGP	0,9022	0,7950	Bayesian approach	0,8470	0,6790
MGP	0,8985	0,7875	Boosting	0,7600	0,7000
2SGP	0,9027	0,8015	Bagging	0,8470	0,6840
GP	0,8889	0,7834	RSM	0,8520	0,6770
Fuzzy classifier	0,8910	0,7940	CCEL	0,8660	0,7460
C4.5	0,8986	0,7773	CART	0,8744	0,7565
LR	0,8696	0,7837	MLP	0,8986	0,7618
k-NN	0,7150	0,7151	GP	0,8960	0,7693
GP RPN	**0,8960**	**0,7550**	**Fuzzy GP**	**0,9010**	**0,7750**

Several real classification problems were solved to compare the algorithm efficiency with analogies. They are:

– Bank clients classification problem, Australian variant, 14 attributes, 2 classes, sample size is 690. Classes and variables meanings are encrypted;

– Bank clients classification problem, German variant, 24 attributes, 2 classes (bad or good customer based on statistical data), sample size if 1000.

For every problem the algorithm was run several times. The average correct classification rates are shown at table 3 (higher is better). The presented algorithm name is Fuzzy GP.

The resulting algorithm has one of the highest efficiencies comparing to competitors. It also is better than the symbolic regression algorithm, which was also tested and has the name GP RPN. Although there are methods with lower error values, they form a symbolic regression which is hard to interpret, while the presented method gives a rules base which is easy to analyze for people.

Conclusions. As a result of this work, a new approach for fuzzy logic systems design was presented. This approach uses evolutionary algorithms with original self-adjustment procedures, which allow automatic choice of most suitable genetic operators. The data representation in genetic programming allows forming small and effective rule bases. The linguistic variables adjustment procedure increases the algorithm efficiency by choosing term positions. All of these advantages give the algorithm an opportunity to compete with the best known algorithms while solving complex classification problems.

References

1. Niehaus J., Banzhaf W. Adaption of Operator Probabilities in Genetic Programming. In: Miller J. et al. (Eds.): EuroGP 2001, LNCS 2038, p. 325–336, 2001.

2. Koza J. Genetic Programming. The MIT Press Cambridge, Massachusetts London, England, 1998.

3. Semenkin E., Semenkina M. Self-Configuring Genetic Programming Algorithm with Modified Uniform Crossover. In: Proc. of IEEE Congress on Evolutionary Computation. IEEE World Congress on Computational Intelligence, Brisbane, Australia, 2012.