

# Systematically Evolving Configuration Parameters for Computational Intelligence Methods

Jason M. Proctor and Rosina Weber

College of Information Science & Technology, Drexel University  
{jrp338, rw37}@drexel.edu

**Abstract.** The configuration of a computational intelligence (CI) method is responsible for its intelligence (e.g. tolerance, flexibility) as well as its accuracy. In this paper, we investigate how to automatically improve the performance of a CI method by finding alternate configuration parameter values that produce more accurate results. We explore this by using a genetic algorithm (GA) to find suitable configurations for the CI methods in an integrated CI system, given several different input data sets. This paper describes the implementation and validation of our approach in the domain of software testing, but ultimately we believe it can be applied in many situations where a CI method must produce accurate results for a wide variety of problems.

## 1 Introduction

Computational intelligence (CI) methods exploit abilities such as tolerance for imprecision, uncertainty, robustness, and partial truth to achieve tractability [11]. Although there is no consensus on the definition of CI, systems that include evolutionary computing, fuzzy computing, and neurocomputing are commonly identified as CI [5, 7].

In order to support their abilities, CI methods are tailored in each application through their configuration. For example, artificial neural networks (ANN) use configuration parameters such as training accuracy and number of epochs, while genetic algorithms (GA) require the specification of a rate of mutation in each generation. These parameters, which are responsible for the flexibility and tolerance of CI methods, are also responsible for their accuracy. In practice, assigning values for these parameters typically relies on suggestions from the literature, which are then adjusted by trial and error. Sometimes, systems allow users to change these values, but usually without providing individualized recommendations. The result is that CI methods typically assume the same default configuration for all executions.

In order to ensure quality, we believe CI methods should incorporate systematic means to find configurations that can produce high accuracy, and thereby recommend an individualized configuration for each input data set. Our approach to systematically finding configurations for a given input set builds on prior tests of that input set. Given the limited availability of test results for multiple configurations as raw data for our analysis, we use a GA to evolutionarily generate and evaluate parameter configurations.

The contributions of this paper are associated with the following questions. First, can individualized configurations for different inputs improve the overall accuracy of a CI method? Second, is there a way to systematically find individualized configurations for inputs to improve accuracy? In the next section we discuss some background to these questions and describe our approach to finding configuration parameters for CI methods. After that, we evaluate these questions by using our approach, and finalize with conclusions and future work.

## 2 The Meta-level Genetic Algorithm (MGA)

### 2.1 Motivation and Background

The potential for a CI method with one fixed parameter configuration to have a lower accuracy than the same method using individualized configurations became a concern in the context of the CI-Tool. The CI-Tool is an integrated suite of CI methods (including ANN, GA, and Info-Fuzzy Networks) for generating test cases for software programs [3, 4, 6, 8]; it must be highly accurate in order to benefit users. The CI-Tool's ANN module is used to analyze the association between inputs and outputs of data-driven programs [8]. The ANN methodology includes construction, training, pruning, and rule-extraction, in order to build a mathematical representation of a target software program (called a testbed) and generate test cases based on that model. These test cases are given as input to the actual testbed program, and the ANN's predicted output for the test cases is compared to the real outcome. The success rate, the ratio of correct test cases to total test cases, represents how well the ANN models the testbed.

The ANN module in the CI-Tool has ten configuration parameters that can be set by the user. An exhaustive evaluation of these would require over fifteen billion processor-years. Clearly, a practical method for finding parameter configurations must be able to converge on a suitable set much faster than that; this is easier if the method can build on past performance. If an automated method for finding suitable configuration values can be developed, then we have the potential to learn more about complex problems and adapt prior solutions to solve them quickly and accurately [9].

In developing and testing the CI-Tool, it was discovered that changing the ANN's configuration parameter values had an apparent effect on accuracy for some testbeds. Detailed exploration of the strength and nature of the effect was hindered by the size of the problem space. Additional evidence of the different impact caused when changing parameter values for different testbeds is given in a study that validated the ANN task with different numbers of training records [2].

Over the last twenty years, many authors have explored many possible interconnections between genetic or evolutionary algorithms and ANNs [1, 10]. Our approach differs from most of these because it only acts indirectly on network weights, architecture, and learning rules, and because it is designed as a reusable component in an integrated CI system, as opposed to a part of the ANN module itself.

## 2.2 Developing the MGA

A genetic algorithm works by exploring the space of input values and evaluating the results to improve the quality of those results in the context of a target task. Because our GA’s target task could be a GA, or any other CI method, we call it a Meta-level GA (MGA). The parameters for the target CI method become the variable space that the MGA explores. The accuracy in which the target CI method performs its original task becomes the reference of fitness for the MGA to evaluate configuration parameters in each improvement step.

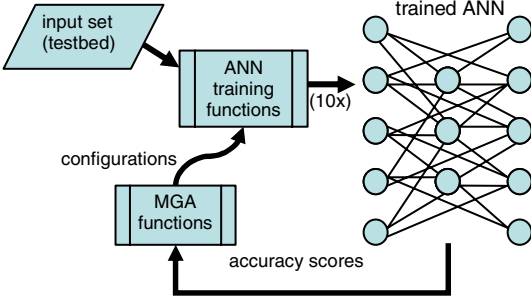


Fig. 1. MGA and ANN

The key functions of the MGA are its fitness function, reproduction function, hybridization function, and mutation function. Most of these could be reused from the CI-Tool’s existing GA module [4], but the implementation required a new fitness evaluation function to execute and interpret the results of the ANN. Figure 1 shows the cyclical interaction of MGA and ANN. Because the

ANN’s initial network weights are seeded randomly, we included a loop to execute each set of configuration parameters ten times for the same training data, and used the average success rate across those iterations as the success rate for that population member. We used the square of the average success rate for breeding selection.

## 3 Evaluation

### 3.1 Methodology

In order to answer our research questions, we executed the MGA to systematically find suitable configurations for the ANN for four input sets. The MGA evolved parameter configurations by modifying the ANN’s parameters and repeatedly testing it using consistent training data. Attempts to simultaneously vary all ten of the ANN’s parameters in the same run caused the MGA not to converge in a reasonable time. Therefore, we studied the data of several test runs and consulted with the module’s designers to identify the best candidates for exploration. The designers suggested three parameters: *training accuracy* was suggested to be the most important indicator of success, followed by *pruning accuracy* and *learning rate*.

The MGA was executed for each testbed with ten iterations per candidate ANN configuration, ten candidates per generation, and at least eight generations. Each iteration produced an accuracy score: the percentage of time the ANN’s model matched the reality of the testbed. The candidate with the best average score was selected for verification. In two of the four cases, the best fitness score was in a generation other than the last. In each of these cases, this parameter configuration was

present in the last generation, but its success rate was slightly lower due to the non-determinism resulting from random seeding of initial network weights in the ANN.

To verify the parameter values, the MGA created twenty more executions of the ANN: ten with the default configuration and ten with the MGA-recommended configuration. To decrease the chance of random error affecting the comparison of results, we used the same training data across all twenty of a testbed's verification runs.

### 3.2 Results

The results show that the MGA approach can significantly improve the average overall accuracy of the ANN task. Table 1 shows the resulting averages for configurations defined by ANN designers (default) and obtained with the MGA for each input, as well as ANOVA significance values. For testbeds 1 and 2, note that the new accuracy is significantly higher at  $p < 0.01$ , while for the others, the new accuracy is slightly lower, but the difference is not statistically significant. For the latter two, the MGA-recommended values were similar to the system defaults.

**Table 1.** Accuracy across four testbeds

	default	MGA	p
Testbed 1	82.2	92.7	0.001
Testbed 2	73.1	90.3	0.004
Testbed 3	90.6	86.8	0.086
Testbed 4	89.2	88.4	0.491
Overall Average	83.8	89.6	0.005

Averaged across all four testbeds, the resulting average accuracy increased from 83.8% to 89.6%;  $F(1, 74) = 8.569$ ,  $p = 0.005$ . This result demonstrates that CI methods can improve their overall accuracy using individualized configurations for different testbeds. It also demonstrates that our proposed MGA can be used as a systematic means to find individual parameter configurations.

## 4 Conclusions and Future Work

### 4.1 Conclusions

Based on the results, we believe the methods typically employed by ANN designers produce results that leave room for improvement. With respect to our first question, we have shown that individualized configuration parameters can improve the overall accuracy of a CI method, making one blanket configuration not ideal for all possible input data sets.

In response to our second question, we conclude that there is a way to systematically find parameters to configure a CI method for a specific input to ensure high accuracy, and that our MGA approach is one way to it. We are not claiming this is the only way or the best way—our approach does not attempt to find optimal

parameters, but a configuration that is at least as good as or better than defaults assigned for all inputs. In fact, as these parameters cannot be optimized, empirical evidence is required to suggest their quality. Our assertion is that one fixed configuration tends to lower overall accuracy, and that it is possible to do better.

It is well known that GAs may get stuck in local maxima, being also susceptible to their own configuration parameter values. Acknowledging this fact, we relied on the designer's experience to guide us through the choice of the ANN parameters we would initially vary. Additionally, we have the accuracy obtained with the default parameters, which although not being a baseline, serve as a valid reference for the GA learning.

## 4.2 Future Work

When using the MGA to find configurations for the CI-Tool's ANN, for practical reasons, we constrained the number of parameters to encourage a workable limit on the time the MGA takes to converge to a suitable configuration. Future work suggests a more flexible stopping strategy, allowing the MGA more time to try to find convergence. The high dimensionality of the problem of configuring the ANN should also be explored more thoroughly.

One possible future step is to determine classes of inputs that are amenable to different configurations so we can recommend these configurations when users submit an input to a CI method. This work suggests deeper methodological ramifications: configurable CI methods should have some kind of self-evaluation (either by MGA or some other method) built in, to improve the quality of systems whenever possible.

ANN is not the only CI method to which the MGA can be applied, although it may be necessary to use some fitness metric other than success rate. Ideally, an MGA could be applied to any CI methods which include their own fitness functions. Because the CI-Tool is an integrated suite of CI method modules, it is an ideal platform for exploring the potential of the MGA on the same set of input problems; this could lead to the ability to recommend one CI method over another.

## Acknowledgements

The authors would like to thank Mr. Tom Barr for his guidance in extending the CI-Tool, and Drs. Mark Last and Abraham Kandel for their suggestions on working with the ANN. This work is supported in part by the National Institute for Systems Test and Productivity at USF under the USA Space and Naval Warfare Systems Command grant no. N00039-02-C-3244, for 2130 032 L0, 2002.

## References

- [1] Abraham, A. and Nath, B., "Hybrid Heuristics for Optimal Design of Neural Nets," in *Developments in Soft Computing: Proceedings of the Third International Conference on Recent Advances in Soft Computing*, R. John and R. Birkenhead, Eds. Berlin: Springer Verlag, 2000, pp. 15–22.

- [2] Agarwal, D., "A Comparative Study of Artificial Neural Networks and Info Fuzzy Networks on Their Use in Software Testing," in *Computer Science & Engineering*. Tampa FL: University of South Florida, 2004.
- [3] Barr, T., "Architectural Overview of the Computational Intelligence Testing Tool," in *Proceedings of the Eighth IEEE International Symposium on High Assurance Systems Engineering*. Los Alamitos CA: IEEE Computer Society, 2004, pp. 269–270.
- [4] Berndt, D., Fisher, J., Johnson, L., Pinglikar, J., and Watkins, A., "Breeding Software Test Cases with Genetic Algorithms," in *Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS '03)*, R. A. Sprague, Jr., Ed. Los Alamitos CA: IEEE Computer Society, 2002.
- [5] Bezdek, J. C., "Computational Intelligence Defined - by Everyone!," in *Computational Intelligence: Soft Computing and Fuzzy-Neuro Integration with Applications*, O. Kaynak, L. A. Zadeh, B. Türksen, and I. J. Rudas, Eds. Berlin: Springer Verlag, 1998, pp. 10–37.
- [6] Last, M., Friedman, M., and Kandel, A., "The Data Mining Approach to Automated Software Testing," in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York NY: ACM Press, 2003, pp. 388–396.
- [7] Pedrycz, W., "Computational Intelligence As an Emerging Paradigm of Software Engineering," in *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering*. New York: ACM Press, 2002, pp. 7–14.
- [8] Saraph, P., Last, M., and Kandel, A., "Test Set Generation and Reduction with Artificial Neural Networks," in *Artificial Intelligence Methods in Software Testing*, M. Last, A. Kandel, and H. Bunke, Eds.: World Scientific, 2004.
- [9] Weber, R., Proctor, J. M., Waldstein, I., and Kriete, A., "CBR for Modeling Complex Systems," in *Proceedings of the Sixth International Conference on Case-Based Reasoning (ICCBR 2005); LNCS 3620*, H. Muñoz-Avila and F. Ricci, Eds. Berlin: Springer (in press), 2005, pp. 625–639.
- [10] Yao, X., "Evolving Artificial Neural Networks," *Proceedings of the IEEE*, vol. 87, pp. 1423–1447, 1999.
- [11] Zadeh, L. A., "Roles of Soft Computing and Fuzzy Logic in the Conception, Design and Deployment of Information/Intelligent systems," in *Computational Intelligence: Soft Computing and Fuzzy-Neuro Integration with Applications*, O. Kaynak, L. A. Zadeh, B. Türksen, and I. J. Rudas, Eds. Berlin: Springer Verlag, 1998, pp. 1–9.