Available online at www.sciencedirect.com
SCIENCE DIRECT®
Fuzzy Sets and Systems 145 (2004) 111–139
FUZZY sets and systems
www.elsevier.com/locate/fss

Building a software experience factory using granular-based models

Marek Reformat[a], Witold Pedrycz[a,b,*], Nicolino Pizzi[c]

[a] *Department of Electrical and Computer Engineering, University of Alberta, Edmonton, Alta., Canada T6G 2G7*
[b] *Systems Research Institute, Polish Academy of Sciences, Warsaw 01-447, Poland*
[c] *Institute for Biodiagnostics, National Research Council Winnipeg, Canada*

Abstract

Software quality is one of the most important practical features of software development. Project managers and developers look for methods and tools supporting software development processes and ensuring a required level of quality. To make such tools relevant, they should provide the designer/manager with some quantitative input useful for purposes of interpretation of the results. Knowledge provided by the tools leads to better understanding of the investigated phenomena. In this paper, we propose a comprehensive development methodology of logic-based models represented by fuzzy neural networks. A process of model development is performed in the stages of structural and parametric optimization. The structural optimization framework utilizes mechanisms of evolutionary computing, which become especially attractive in light of the structural optimization of the models. The parametric optimization is performed using the gradient-base method. The study comprises two detailed case studies dealing with existing software data. The first one deals with the quality assessment of software objects in an exploratory biomedical data analysis and visualization system. The second case is concerned with the model of software development effort discussed in the setting of some medical information system.

Keywords: Granular computing; Logic-based models; Software experience factory; Software quality; Software experimental data; Evolutionary computing; Software development effort

1. Introduction

Software quality is one of the most important topics related to software development. Project managers and developers look for methods and tools supporting software development processes and ensuring a required level of quality. However, software is different from most products. First, software

* Corresponding author. Department of Electrical and Computer Engineering, University of Alberta, Edmonton, Alta., Canada T6G 2G7. Tel.: +1-204-474-8380; fax: +1-204-261-4639.
E-mail address: pedrycz@ee.ualberta.ca (W. Pedrycz).

0165-0114/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/j.fss.2003.10.007

is developed rather than produced. Second, software has a non-visible nature—it is difficult to see the structure or the function of software. This very nature of software means that a special attention is needed to ensure development of high quality systems. An interesting approach addressing this problem is the Experience Factory [3]. It is "a logical and/or physical organization that supports project developments by analyzing and synthesizing all kinds of experience, acting as a repository for such experience, and supplying that experience to various projects on demand". Gained experiences are deployed into recommended best practices, estimation models, and software development training courses, which are spread as process improvements throughout development teams. The main idea behind Experience Factory is reuse. In this case reuse is related not only to software products, components and objects, but mainly to experience and knowledge that was gained during previous software development processes. In this case, knowledge has to be packaged appropriately for reuse.

The paradigm of the Experience Factory identifies a need for collecting software data, analysis of the data and representation of results obtained from the analysis. The Factory describes creation of an experience database and defines a basic framework for these activities. However, it does not indicate methods and approaches that are to be used.

The first phase of constructing the Factory is gathering data that represent and describe software products and processes related to software development. Collected data have to be in a format that allows automatic/semi-automatic processing. In order to fulfill this requirement, software modules and objects are represented using software metrics. This approach has already been identified as one of the ways of dealing with qualitative problems in a quantitative way [2,4]. Software metrics can be extracted directly from software and illustrates different features and attributes of objects. Measures can also be identified features that are not directly extracted from the software and related to processes such as construction, testing, maintenance, and even evolution of software objects. In this case, measures represent features of objects such as the number of defects, object readability, and compliance with coding standards. Having such a set of data available, one can look at different methods and approaches to process the data and construct models representing relationships existing between different measures of software objects. These activities represent the second stage of building the Experience Factory and are the main subject of this study.

Knowledge extraction from data and its simultaneous processing can be accomplished by synthesizing a class of models capable of dealing with two highly conflicting design requirements such as transparency and accuracy of a model. In an attempt to address these issues in our study, we note the following:

- the concept of granular computation is extremely promising, granules of software measures can be used in analysis of software processes, as well as in the development of models with the intent of achieving a high level of comprehension of phenomena represented by the data;
- models of well-defined logic structure are essential, this becomes a prerequisite to their transparency and user-friendliness;
- a multiphase process in which we clearly delineate between structural and parametric development is essential; the process starts with building a "skeleton" (more qualitative than quantitative) or blueprint of the model that is concerned with the structural relationships, and subsequently, we concentrate on its further numeric refinement (parametric optimization).

The set of observations mentioned above, has led to the application of fuzzy neural networks as the best "candidates" for transparent yet accurate models built on principles of granular computing.

Finding the best structure of the model is accomplished by application of evolutionary computing techniques.

The proposed approach is used to design models representing processes of quality assessment of software objects and estimation of development efforts of software modules. Both processes are important elements of the Experience Factory. Models of these processes are simultaneously "packages" of knowledge and estimation tools. They can be used to guide the development process as well as evaluate software modules and objects.

The paper is organized into 6 sections. Section 2 introduces basic concepts of granulation and granular-based models. Section 3 describes evolutionary-based techniques treated as a design backbone of such models. Section 4 covers details of data analysis and an overall modeling of the process of quality assessment. In Section 5, analysis and modeling approaches are applied to a process of estimating development efforts of software modules. Section 6 presents concluding remarks.

2. Granular computation

2.1. Information granulation

By its very nature, software data are complex to analyze. There are several main reasons, namely (a) nonlinear nature of relationships existing between experimental data, (b) a need for visualization of results to support their understanding and interpretation, and (c) scarcity of data combined with a diversity of additional factors not conveyed by the dataset itself.

While the use of statistical methods is pervasive in data analysis, the scope of such analysis can be enhanced by the technology of granular computing. Granular computing is geared toward representing and processing information granules. Information granules are collections of entities, usually originating at the numeric level, that are arranged together due to their similarity, functional adjacency, indistinguishability or alike [13]. This approach is suitable in situations when there is a need to comprehend the problem and provide greater insight into its essence rather than being inundated by unnecessary details. In this sense, granulation serves as an important mechanism of abstraction. This approach allows hiding some imperfections in analyzed data thereby making the analysis more immune to outliers.

Granular computation often exploits fuzzy set theory to represent knowledge in terms of information granules. Fuzzy sets are particularly advantageous due to their clearly defined semantics, which can formalize the linguistic terms used by users/designers when analyzing data. One can easily change the size of information granules (by adjusting the membership functions) so that subsequent data analysis may be performed at the most beneficial and parsimonious level of detail.

2.2. Granular-based system

Representation of data using information granules leads to the application of models that are both capable of dealing with granules and expressive of their interrelationships. Fuzzy modeling, which hinges on information granules, can generate such models, while fuzzy sets and fuzzy relations can represent the granules. The interaction of the models with the environment (predominantly numeric) is realized through model interfaces. The input interface (encoder or fuzzification block)

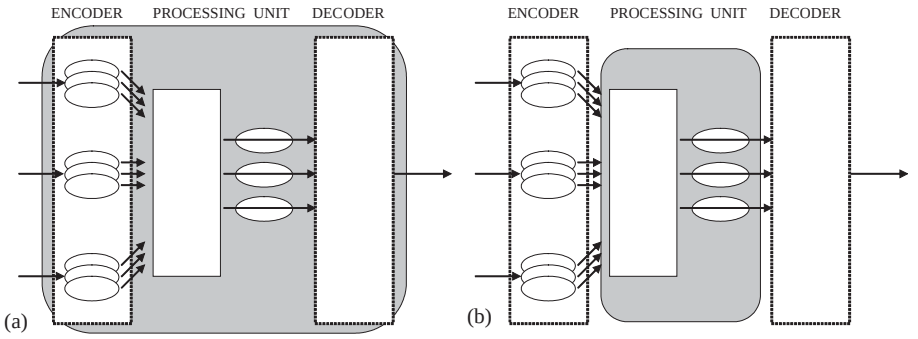

Fig. 1. Two fundamental scenarios of the development of fuzzy models (gray boxes represent areas of modeling activity): external optimization at the level of numeric data (a) and internal optimization at the level of fuzzy sets in the input and output space (b).

expresses input data in the format compatible with fuzzy set granules. The output interface (decoder or defuzzification module) translates the results, derived from the appropriate level of granularity, into the numeric format of the modeling environment (where the original numeric data come from).

As discussed in [12], two main modeling and ensuing development scenarios can be envisioned, refer to Fig. 1.

In the first scenario (Fig. 1(a)), we are concerned with the external optimization of the fuzzy model viz. a way in which it manifests at the level of numeric experimental data. From the functional standpoint, the fuzzy model is a structure of three mappings put in series, that is the encoder $\mathbf{x} = ENC(\boldsymbol{x})$, the processing part of the fuzzy model, $\mathbf{y} = FM(\mathbf{x}, \mathbf{p})$, and the decoder $\boldsymbol{y} = DEC(\mathbf{y})$ where $\mathbf{x}$ and $\mathbf{y}$ are elements in the unit hypercube while "$\boldsymbol{x}$" and "$\boldsymbol{y}$" are the respective input and output of the model manifested at the numeric level, that is $\boldsymbol{x}$, $\boldsymbol{y}$ are reals. As the experimental data are given in the form of input–output numeric pairs, say $\{\boldsymbol{x}(k), \boldsymbol{target}(k)\}$, $k = 1, 2, \ldots, N$, and we require that $\boldsymbol{y}(k)$ (the output of the fuzzy model for the input $\boldsymbol{x}(k)$) equal $target(k)$, $\boldsymbol{y}(k) \approx \boldsymbol{target}(k)$, the optimization of the parameters of the fuzzy model ($\mathbf{p}$) requires a backpropagation of the error through the decoder. In other words, the gradient-based scheme requires the calculations of the following expression

$$\frac{\partial y(k)}{\partial p_i} = \frac{\partial DEC(\mathbf{y})}{\partial FM(\mathbf{x}, \mathbf{p})} \frac{\partial FM(\mathbf{x}, \mathbf{p})}{\partial p_i}$$

where "i" denotes the ith position of the vector of the parameters of the fuzzy model. It becomes evident that the form of the decoder plays an important role in the optimization process because of the chaining effect i.e., a way in which the parameters of the model ($\mathbf{p}$) can be "accessed" for optimization purposes. Furthermore as both the encoder and decoder are included in the model identification, they may be optimized as well.

While the first scenario could be viewed an *external* mode of fuzzy identification, the second one is an internal one. As shown in Fig. 1(b), the processing module is optimized here. The original training data $\{\boldsymbol{x}(k), target(k)\}$ are converted through fuzzy sets occurring in the encoding and decoding blocks $\{\mathbf{x}(k), \boldsymbol{target}(k)\}$ and as such used for the optimization of the fuzzy model. We consider these fuzzy sets to be fixed in advance. In other words, we end up with a requirement

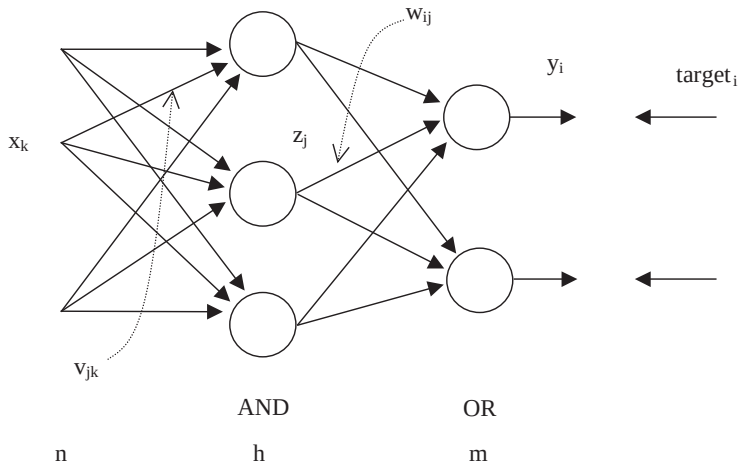

Fig. 2. A structure of the fuzzy neural network along with detailed notation.

$\mathbf{y}(k) \approx \mathbf{target}(k)$ $k = 1, 2, \ldots, N$. More realistically, a sum of squared errors needs to be minimized with regard to the structure of the fuzzy model (FM) and its parameters,

$$Q = \sum_{k=1}^{N} (FM(\boldsymbol{x}(k), \boldsymbol{p}) - \boldsymbol{target}(k))^{\mathrm{T}} (FM(\boldsymbol{x}(k), \boldsymbol{p}) - \boldsymbol{target}(k)) \Rightarrow \text{Min}.$$

In this study, we concentrate on granular modeling, which is geared toward internal optimization, Fig. 1b. This type of optimization promotes the interpretability of the fuzzy model as we focus on the mapping of the fuzzy sets rather than the numeric experimental data.

The structure of the fuzzy neural network, Fig. 2, is fully determined by the structural optimization. The gradient-based learning of the network leads to further refinements that appear at the numeric level. Proceeding with the architectural details, the fuzzy neural network is governed by the following expressions (refer to Fig. 2).

As we noted, the mapping from the structure to the fuzzy neural network is straightforward. Recall that an h-input single output OR neuron is described in the form

$$y = OR(\mathbf{z}; \mathbf{w})$$

where $\mathbf{z}$, $y \in [0, 1]$. The connections $w_1, w_2, \ldots, w_h$ are arranged in a vector form ($\mathbf{w}$). Rewriting the above expression in a coordinate-wise manner, we obtain

$$y = \mathop{\mathbf{S}}_{i=1}^{h} (z_i t w_i)$$

where 't' means t-norm modeling logical connection AND, and 's' is a t-conorm representing logical connection OR. The above expression means that the neuron realizes an s–t composition of the corresponding finite sets $\mathbf{z}$ and $\mathbf{w}$.

Table 1
A learning environment of the fuzzy neural network: specification of the problem and an on-line learning algorithm

On-line learning mode:
(updates of the connections are preformed after processing of each input–output pair of data: {**x** **target**})
Performance index: $Q = \Sigma_k(\mathbf{y}(\mathbf{k}) - \mathbf{target}(\mathbf{k}))^T(\mathbf{y}(\mathbf{k}) - \mathbf{target}(\mathbf{k}))$
(where y(k) is the output of the network for a data point k)
Initial connections: random initialization
Update formulas for the connections
$w_{ij}(iter+) = w_{ij}(iter) - \alpha \dfrac{\partial Q}{\partial w_{ij}(iter)}$
$v_{jk}(iter+) = v_{jk}(iter) - \alpha \dfrac{\partial Q}{\partial v_{jk}(iter)}$

The AND neuron $z = AND(\mathbf{x}; \mathbf{v})$ is governed by the expression

$$z = \mathop{\mathbf{T}}_{i=1}^{n}(x_i s v_i)$$

with 't' and 's' defined as above. Computationally, this neuron realizes a standard t–s composition of **x** and **v**.

The role of the connections in both neurons is to weight the inputs in order to provide sufficient parametric flexibility. A monotonicity property holds. In case of OR neurons, the higher the connection weight, the more essential the associated input. For AND neurons an opposite situation holds: a lower connection weight indicates that the respective input is more essential. In general, a certain thresholding operation can be sought. For any OR neuron, we consider the input irrelevant if the associated connection weight assumes values lower than 0.5. An input of the AND neuron is viewed irrelevant if the connection weight exceeds 0.5.

The learning is realized as a gradient-based optimization scheme [11]. The details of the on-line learning algorithm are given in Table 1. The listed computations use the t-norm and t-conorm specified as a product operation and a probabilistic sum. The choice of these intuitively simple (co)norms has been motivated by the fact that they are commonly used in many implementations of fuzzy neural networks. However, they lead to problems related to data dimensionality (see below).

The parametric learning of the fuzzy neural network has been well developed and documented in the literature [11,14]. Several general observations are worth summarizing:

- The gradient-based learning supports optimization that may result in a local minimum of the performance index. Global minimum could be out of reach of this learning mechanism.
- The efficiency of learning depends upon the choice of the triangular norms and co-norms. Here the minimum and maximum operators deserve particular attention as they lead to optimization traps. One of the reasons is that both minimum and maximum are non-interactive meaning that the results depend upon extreme values encountered and the final outcome does not reflect the remaining arguments of these t- and s-norms. On the other hand, for most other t-norms, we

may end up with a saturation effect that may be extremely difficult to handle in case of higher dimensions of the problem. For instance, consider the product as a model of the t-norm. If the number of arguments increases, the result of aggregation carried out in this way tends to zero. Now, if one envisions such an AND neuron located in the input layer of the fuzzy neural network and assume that all connections are the same and equal to zero, the output of the neuron reads as $z = \prod_{i=1}^{n} x_i$. For any input less than one, say $1 - \gamma$ we end up with the output equal to $(1 - \gamma)^n$. One can easily check that a situation in which $\gamma = 0.5$ and $n = 40$ inputs produces the output of the neuron equal to $9.095 * 10^{-13}$. This activation level reduces quickly once the dimensionality of the problem goes up.

- The learning may be very slow especially when the size of the network gets large. The way in which the connections are initialized (random values) precludes any initial associations to preliminary knowledge about the network structure (this implies a fully connected topology where all neurons are connected with the neurons in neighboring layers), nor it can guard against the curse of dimensionality.

In light of these observations, the general design paradigm proposed in this study is strongly supported. Instead of learning the fuzzy neural network from scratch (a process that may fail quite easily), we concentrate first on establishing a structural blueprint of the network and then continue with the learning of the connections. In the approach proposed, the structural optimization means selection of a given (usually small) number of connections between input and hidden nodes, as well as between hidden and output nodes. Effectively, this network blueprint reduces the number of connections to be learned and practically eliminates the saturation effect (see above). Structural optimization of the network cannot be achieved through parametric (gradient-based) optimization and requires methods along the line of evolutionary computing [8,20].

3. Evolutionary-based synthesis of granular models

3.1. Concept

The evident diversity of fuzzy models existing in the literature is amazingly uniform in terms of their underlying design. In a nutshell, the original numeric data are transformed through fuzzy sets and this is followed by the construction (estimation) of the parameters of the fuzzy model.

In contrast, the proposed development of the fuzzy model comprises three fundamental phases:

- First, we produce a collection of fuzzy sets (or fuzzy relations) that may be regarded as basic conceptual "building" blocks of the fuzzy model. We would like these entities to be as independent as possible from the ensuing detailed structure of the model. In this sense, these information granules stand on their own and exhibit a well-defined semantics.
- Second, we look for a structure of the fuzzy model. Any structural optimization of the model is far more challenging than a parametric optimization (adjustment of the values of the parameters). At this phase, we exploit evolutionary computing to optimize the structure. Furthermore, the structural optimization is carried out independently from parametric optimization. By distinguishing between the structure and the parameters, we attempt to concentrate on the topology of the model and

make it disjoint (as much as possible) from the phase concentrated on parameter adjustment. The best structure is found by searching the space of all possible structures and, subsequently, proceed with fine tuning at the parametric (numeric) level. Evolutionary techniques are essential because structural optimization is unattainable via gradient-based methods and the enormous space of possible structures may only be adequately traversed using evolutionary techniques. As to the structure itself, we proceed with a standard two-level OR–AND representation of functions of symbols (in this phase, fuzzy sets are used as symbols). Interestingly, this representation is in line with the well-known structures of rules (if–then statements) composed of fuzzy sets standing in their condition and conclusion parts.

- Third, once the topology of the network has been established during the previous phase, the network is subject to some parametric refinement. To make this process possible, the network is augmented by modifiable connections and this gives rise to the notion of fuzzy neural networks [12]. In these networks, two types of processing units (fuzzy neurons) are encountered. An OR neuron generalizes an *or*-type of aggregation and an *and*-type of aggregation is realized by using an AND fuzzy neuron. The connections of these neurons help calibrate the inputs and contribute to the improved performance of the model expressed at the level of the information granules (fuzzy sets) now being treated at the numeric end.

3.2. Evolutionary computation

The main part of the proposed three-phase development methodology is structural optimization of the network. As it has been stated above, evolutionary methods are proposed to perform this task. Since its initiation in seventies, the Evolutionary Computation (EC) has become more diverse and a number of different approaches and algorithms have been proposed. A number of different techniques has been developed and applied to optimization problems in different areas of science and engineering [1,5,6]. It seems that two most popular algorithms of EC are Genetic Algorithms (GA) [6] and Genetic Programming (GP) [9]. Because of that, an illustration of both these techniques is presented in the paper. It should be stressed that GA as well as GP are both suitable and capable of solving the problem of network optimization.

After a short and generic description of the concept of the evolutionary algorithms, GA and GP are described. The way they have been "adopted" to the structural optimization of the network is also presented.

The Evolutionary Algorithms (EA) are search methods utilizing the principles of natural selection and genetics [7]. In general, EA operate on a set of potential solutions to a given problem. These solutions are encoded into chromosome-like data structures named genotypes. A set of genotypes is called a population. The genotypes are evaluated based on their ability to solve the problem represented by a fitness function. The results of the evaluation are used in a process of forming a new set of potential solutions. The choice of individuals to be reproduced into the next population is performed in a process called selection. This process is based on the fitness values assigned to each genotype. Genetic operators, i.e. crossover and mutation [18], are employed to form a new population. Crossover allows an exchange of information among individual genotypes in the population and provides innovative capability to the EA. Mutation ensures the diversity needed in the evolution. A sequence of such actions is repeated until some final criterion is fulfilled.

3.2.1. Genetic algorithms

GA are one of the most popular methods of evolutionary computation. An implementation of a typical GA begins with the creation of an initial population, which consists of genotypes generated at random. Next, an evaluation of genotypes from an initial population is performed. Each genotype string is broken into substrings that are used as the input values to the fitness function. The number of substrings depends on the problem to be solved and is unlimited. The fitness value is assigned to each genotype based on the value of fitness function obtained for a given genotype. An intermediate population is created next. This population is the result of a selection process.

The selection can be performed in numerous ways. One of them is called stochastic sampling with replacement. In this method, the entire population is mapped onto a roulette wheel where each individual is represented by the area corresponding to its fitness. Individuals of an intermediate population are chosen by repetitive spinning of the roulette wheel.

Finally, the operations of crossover and mutation are performed on individuals from intermediate population. This process leads to the creation of the next population. Crossover is applied to the randomly paired genotypes from the intermediate population with the probability p_c. In the case of a basic crossover, called one-point crossover, the genotypes of each pair are split into two parts at the crossover point generated at random, and then recombined into a new pair of genotypes. In the case of mutation, all bits in all substrings of genotypes are altered with the probability p_m.

All steps described above are then repeated. The population being evaluated is named the current population and is the same as the next population from the previous iteration (each iteration is referred to as a generation). GA pseudo-code is presented below where $P_{\mathrm{GA}}(t_{\mathrm{GA}})$ denotes the GA population at t_{GA} generation.

Genetic algorithm
$t_{\mathrm{GA}} := 0$
initialize $P_{\mathrm{GA}}(t_{\mathrm{GA}})$
evaluate $P_{\mathrm{GA}}(t_{\mathrm{GA}})$
while not terminate **do**
  $P'_{\mathrm{GA}}(t_{\mathrm{GA}}) :=$ select $(P_{\mathrm{GA}}(t_{\mathrm{GA}}))$;
  $P''_{\mathrm{GA}}(t_{\mathrm{GA}}) :=$ crossover $(P'_{\mathrm{GA}}(t_{\mathrm{GA}}))$;
  $P'''_{\mathrm{GA}}(t_{\mathrm{GA}}) :=$ mutate $(P''_{\mathrm{GA}}(t_{\mathrm{GA}}))$;
  $P_{\mathrm{GA}}(t_{\mathrm{GA}}+1) := P'''_{\mathrm{GA}}(t_{\mathrm{GA}})$;
  $t_{\mathrm{GA}} := t_{\mathrm{GA}} + 1$;
  evaluate $P_{\mathrm{GA}}(t_{\mathrm{GA}})$
od

3.2.2. Genetic programming

Another evolutionary algorithm is GP. GP can be seen as an extension of the genetic paradigm into the area of programs. Objects, which constitute the population, are not fixed-length strings that encode possible solutions to the given problem, but they are programs, which "are" candidate solutions to the problem. In general, these programs are expressed as parse trees, rather than as lines of code. A tree representation of a simple program (instruction) "$\mathbf{a} + \mathbf{b}^{*}\mathbf{c}$" is shown in Fig. 3.

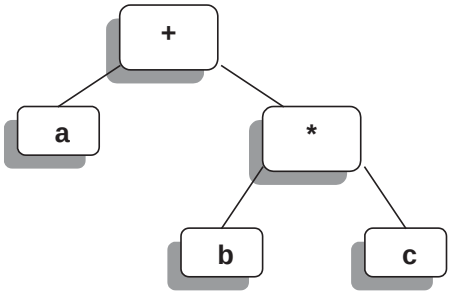

Fig. 3. Tree representation of the instruction "**a** + **b** ∗ **c**".

This representation of candidate solutions combined with some constraints regarding their structure allows for a straightforward representation of different structures. Each structure is evaluated by solving a given problem. After the evaluation a selection is performed. In this process, individuals are chosen to form the next population. The choice is made on the basis of favoring individuals with higher fitness values [1,6]. Crossover and mutation are two operations, which modify structures performing manipulations on the lists [9]. This results in overall "improvement" of individuals from one generation to the next.

3.3. Genetic algorithms-based construction of models

Having identified the shortcomings of learning in the fuzzy model, an intuitively appealing approach has been proposed. It relays on a hybrid learning methodology comprising of two fundamental techniques, namely genetic optimization and gradient-based optimization.

To battle the dimensionality problem, a skeleton of the network is constructed by detecting the most essential connections that shape the architecture and then the detailed optimization of these connections is performed. As to the blueprint, the network has to capture the essence of the mapping of an input to an output. Therefore, the connections are binary (0–1) for the purpose of this optimization. The optimization has to be global. In light of these two requirements, genetic optimization is a suitable avenue to follow.

A standard version of a GA is considered first as an optimization method used for structural optimization. The structure of the network is represented by a binary string. As the connections are two-valued, the entries of the string map directly onto the connections of the neurons. The arrangement of the connections is presented in Fig. 4, which illustrates the details of the structure of the network pertain to the data with 36 inputs (12 input variables with 3 fuzzy sets per variable) 3 outputs (that is, 3 fuzzy sets defined for the output variable) and connectivity of 2 inputs per node in the hidden layer.

The treatment of the connections as binary entities is highly desirable. By doing this, we concentrate our effort on the determination of the most essential structure of the model that attempts to reveal the crux of its topology. We anticipate that such structure will hold in spite of small variations in data and therefore could be regarded as a Boolean "skeleton" of the processing core of the model.

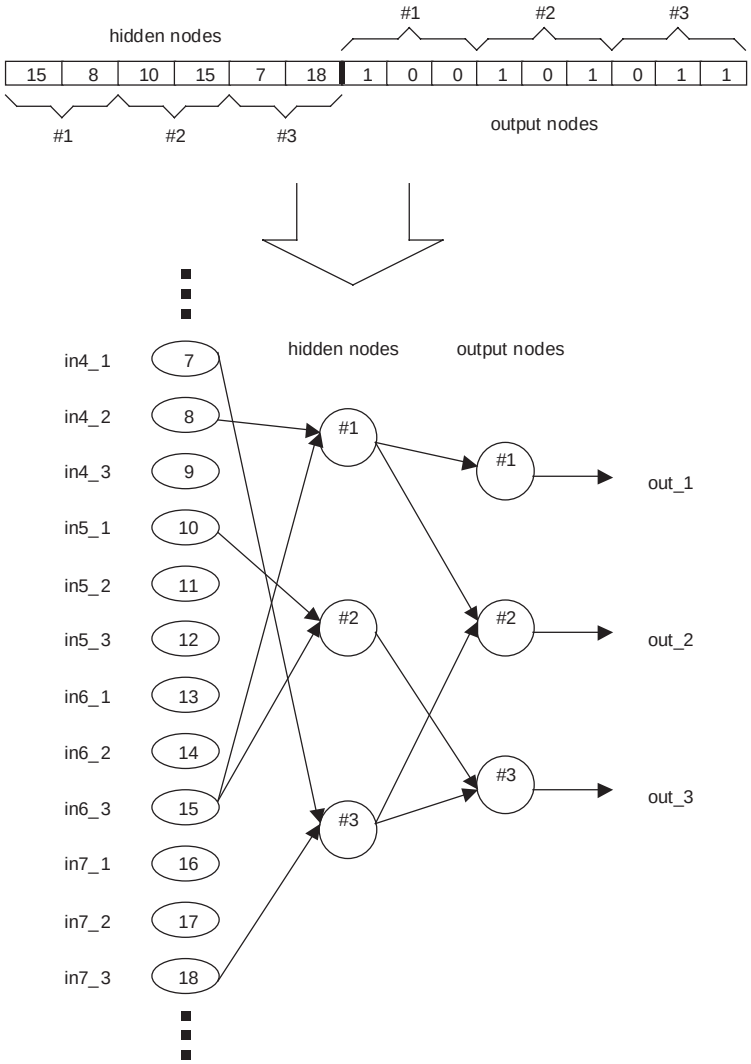

Fig. 4. A part of the network and its one-to-one correspondence with the binary string to be optimized by GA.

The binary connections of the neuron have a straightforward interpretation. In the case of the AND neuron: if the connection is equal to zero, this means that the corresponding input impacts the output of this neuron. For the OR neuron, the converse holds: a connection equal to one implies that the specific input is essential and affects the output of the neuron.

From the design standpoint, we can control the development of the topology of the network. In particular, we can fix the level of connectivity between the input and hidden layer. We may refer to a maximal number of connections from the input nodes to each node (AND neuron) in the hidden layer as the network *connectivity*. For instance, if we have two inputs to the first AND neuron, four to the second and three to the third, we say that the connectivity of the neuron is equal four.

This connectivity analysis offers some insight into the nature of the data. First, the number of meaningful (zero) connections of the neuron determines a local dimensionality of the input space; the contributing inputs form this *local* space. The dimensionality of this space can vary from neuron to neuron. By varying the maximum number of connections that are allowed, we can better realize the intrinsic dimensionality of the data space. The connections of the OR neuron indicate which of these local input spaces contribute to the output of the network.

An initial configuration of the network skeleton is set up as follows. For the given number of neurons in the hidden layer and the connectivity level (that is the number of the inputs going to each neuron in the hidden layer), we randomly initialize the connections of the AND neurons (making sure that the number of the connections between the inputs and each neuron is as specified). Initially, the connections of the OR neurons are all equal to one. Then we allow the genetic optimization to adjust the connections. The adjustment of connections is driven by the fitness function, which is an objective function of an optimization process. The adjustments are realized by the two genetic operations: crossover and mutation. Details of these elements are as follow:

- *Fitness function*
  The role of the fitness function is to assess how well the model matches the experimental data. We are concerned with fuzzy sets defined in the input spaces. While they show up in the lists as symbols (say, A_1, B_3, etc.), their membership functions are present in the computations of the fitness function. The fitness function is computed in the form:

$$Fit\_Fun = \frac{1}{1+Q},$$

  where Q is a commonly used performance index assuming the form

$$Q = \sum_{k=1}^{N} (\mathbf{F}(k) - \hat{\mathbf{F}}(k))^{\mathrm{T}} (\mathbf{F}(k) - \hat{\mathbf{F}}(k))$$

  with N being the number of training data points

$$\mathbf{F}(k) = [F_1(k), F_2(k), \ldots, F_m(k)]^{\mathrm{T}},$$

$$\hat{\mathbf{F}}(k) = [F_1(k), F_2(k), \ldots, F_m(k)]^{\mathrm{T}},$$

  $F_i(k)$ the value of the output "i" for given training data point "k" obtained from the model; $\hat{F}_i(k)$ the original value of the output "i" for given training data point "k"; m the number of outputs, equal to number of fuzzy sets defined in the output space
- *Crossover*
  The role of this operator is to explore the search space formed by a collection of lists. The mechanism of a possible crossover is presented in Fig. 5. An arbitrary point in the parent strings is picked and the information is exchanged between parents. As a result, two new strings are obtained that have beginnings and ends coming from different parent strings.

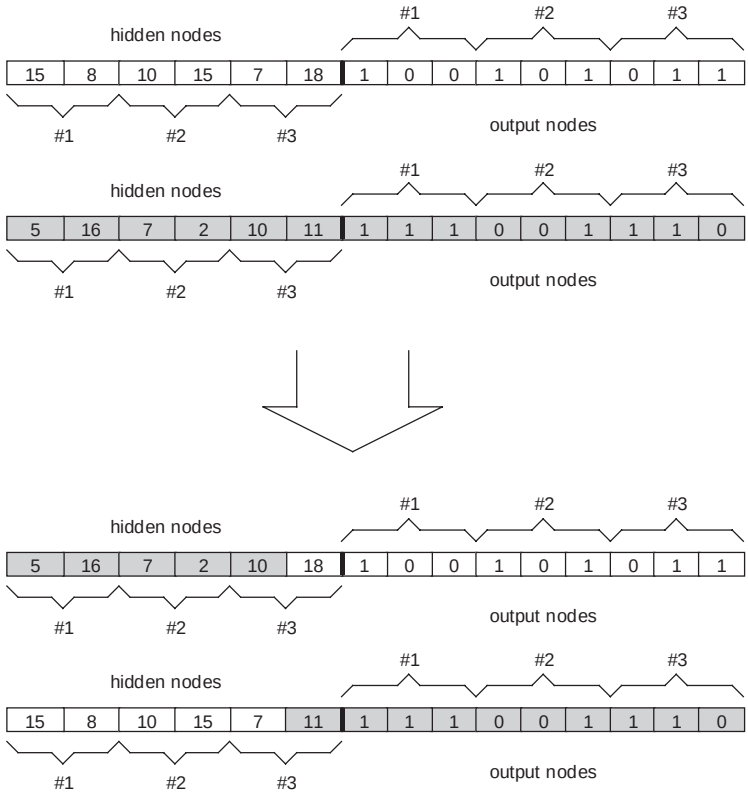

Fig. 5. Crossover operation used in the genetic optimization of the model.

- *Mutation*
  The role of mutation is to introduce diversity into a population and promote a probabilistic-like traversing of the search space. As shown in Fig. 6, mutations form a new binary string by making an alteration to the string in a copy of a single, parent string.

3.4. Genetic programming-based construction of models

The second approach to evolutionary-based construction of models uses GP as an optimization tool. Differences of GP, in comparison to GA, result in different and more flexible representations of the constructed models. GP operates on a population of lists, which are blueprints of fuzzy models. In other words, each individual of a population—a list—represents a single fuzzy model. A fuzzy neural network of single output is a tree with an OR node as the root, AND nodes at the first level, and nodes representing inputs at the second level (see Fig. 7). The OR and AND nodes can have multiple inputs. Additionally, in order to represent fuzzy neural networks with multiple outputs, a single AND node can be connected to more than one OR node.

A population of fuzzy models evolves according to the rules of selection and genetic operations such as crossover and mutation. Each individual in the population (Fig. 8) is evaluated by means

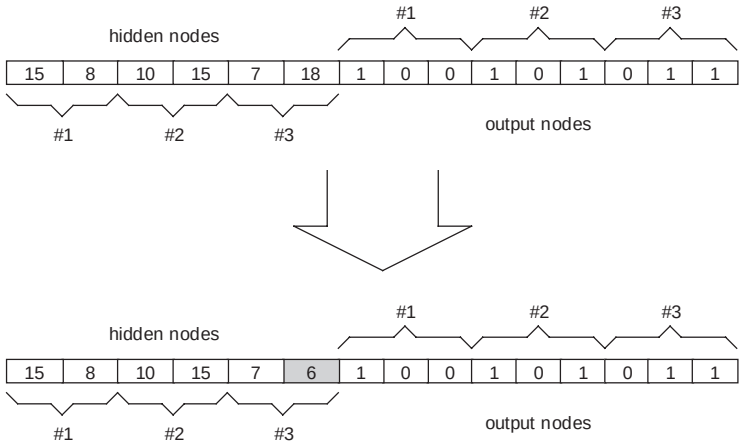

Fig. 6. Mutation operation used in the genetic optimization of the model.

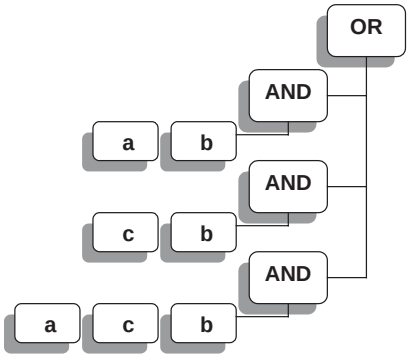

Fig. 7. Tree representation of a simple fuzzy neural network.

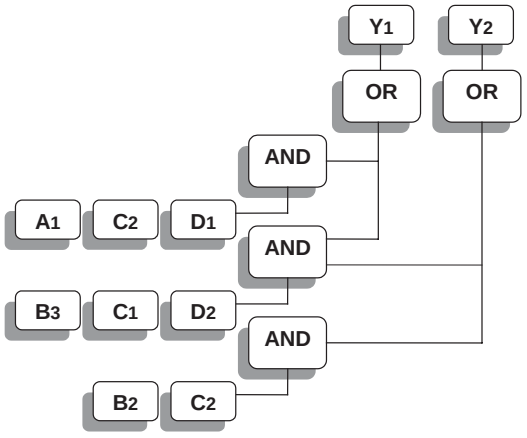

Fig. 8. Single list representation of the fuzzy neural network. The fuzzy sets, A_2, B_2, C_1, C_2, D_1, D_2, are defined using the respective inputs (input spaces), $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{D}$. The fuzzy sets, Y_1, Y_2, are defined using the output space $\mathbf{Y}$.

of a certain fitness function. In this process, individuals are selected to form the next population on the basis of favoring individuals with higher fitness values [1,6,7,9].

Crossover and mutation are two operations that modify the structure of fuzzy models performing manipulations on the lists [19]. This results in "improvement" of population from generation to generation. Let us discuss these components in detail.

- *Fitness function*
  The fitness function represents an objective of the optimization process. The GP-based construction process uses the same fitness function as the GA-based process, i.e.

$$Fit\_Fun = \frac{1}{1+Q},$$

  where Q is a commonly used performance index representing the error between the output of the network and the outputs of the training data points, its form is

$$Q = \sum_{k=1}^{N} (\mathbf{F}(k) - \hat{\mathbf{F}}(k))^{\mathrm{T}} (\mathbf{F}(k) - \hat{\mathbf{F}}(k))$$

  (the description of used symbols is in Section 3.3)
- *Crossover*
  The proposed GP approach uses two crossover operations. One of them is performed at the level of inputs to OR node. In other words, it means that this crossover leads to exchange of number of AND nodes. The mechanism of the crossover is presented in Fig. 9a. Two arbitrary points in parent structures A and B are picked. A structure from parent A is copied from the root down to the crossover point, then a structure from parent B is copied from the crossover point to the bottom of the structure. The new structure is a concatenation of a "head" of one parent structure with a "tail" of the other.

  The second crossover operation is performed at the level of AND node inputs. In this case an exchange process is restricted only to inputs to an AND node. The crossover is illustrated in Fig. 9b. As it can be seen, a set of inputs from a randomly selected AND node replaces a set of inputs of a randomly selected AND node from the other parent.

  Using two crossover mechanisms we have introduced different levels of alternation of parents in a process of creation of offspring. By its nature a GP crossover creates a different offspring created as the result of crossover of two identical parents. This can have a disruptive effect as well as be useful in a process of searching of the domain.
- *Mutation*
  The GP mutation is shown in Fig. 10, mutations build new structures by making (usually small) alterations to the string in a copy of a single, parent structure. Also in the case of mutation we can have two kinds of mutation operations. The one which is performed on the level of AND nodes, Fig. 10a, and the other performed on the level of inputs to a single AND node, Fig. 10b.

An important step in GP is the generation of an initial population of strings (structures). All lists representing fuzzy models are generated randomly with the constraint that each AND node cannot have more than one fuzzy set defined for the same variable. The interpretation aspect of the ensuing model motivates the introduction of this constraint as any eventual duplicates are not easily

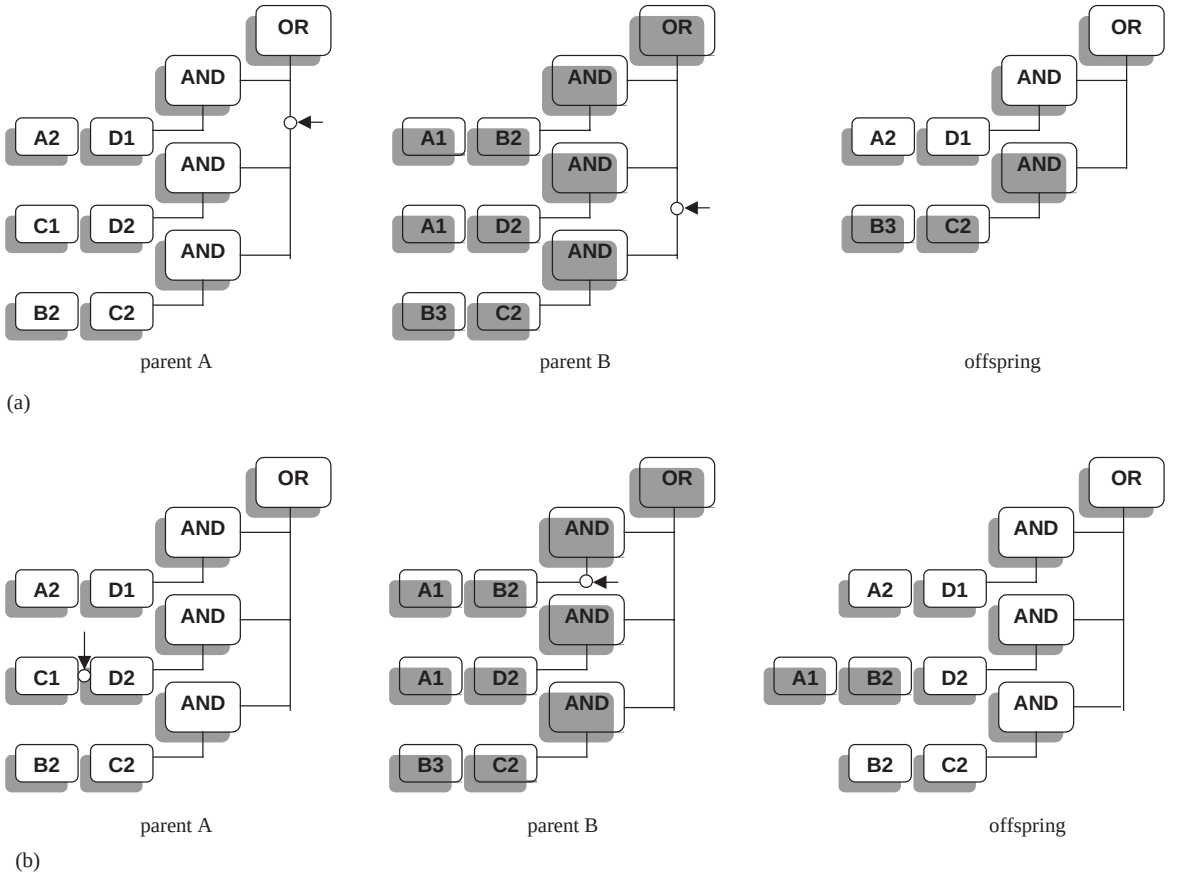

Fig. 9. Crossover operation ('o' indicates a crossover point). Crossover can be done at the level of AND nodes (a), or at the level of rules (b).

interpretable. The same approach is used for the mutation operation, where new strings of inputs are created (Fig. 10). In the case of crossover, we carry out checking for a repetition of fuzzy sets occurring in strings of the same rule obtained during the exchange process (Fig. 9b).

4. Quality assessment of software objects

A primary aim of the Experience Factory is to support activities that lead assurances of software quality. Gathering experience related to these activities and finding ways to simulate and perform them is an important element of the Factory. The presented approach of a granular-based model is applied to software quality data obtained during a process of quality evaluation. The developed model, which at the same time represents knowledge related to software quality evaluation, becomes an essential part of the Factory. The model can also serve as a "quality filter" for software objects. Using it, a system manager can evaluate quality of software objects and identify low quality objects for review and possible revision.

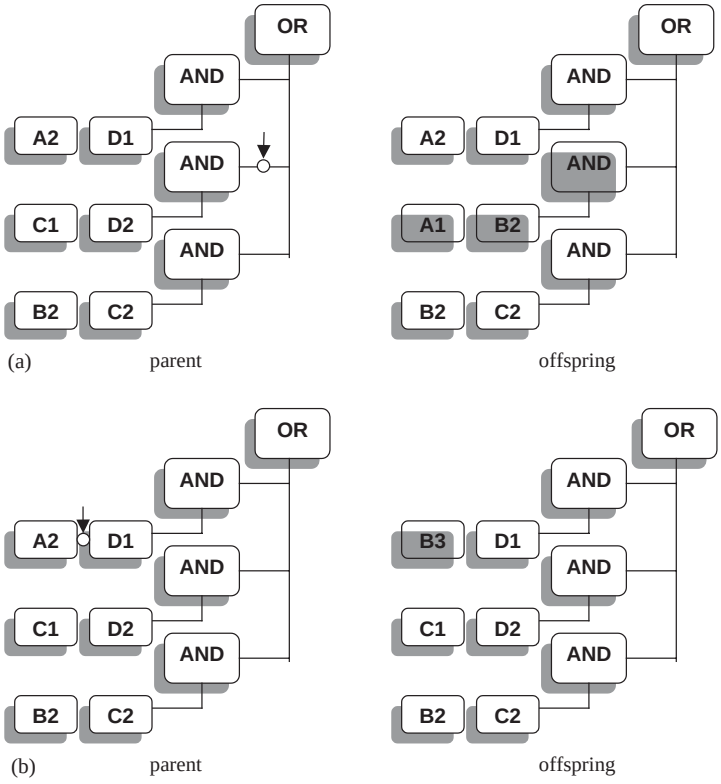

Fig. 10. Mutation operation: 'o' indicates where a newly generated string is introduced—at the level of an AND node (a) or at the level of a rule (b).

The quality of a software system depends on the quality of its underlying individual software objects, while the quality of those objects can be related to a number of attributes. An experiment has been performed to generate data needed for the illustration of the efficacy of the proposed approach to support quality analysis. In the experiment, objects of the system EvIdent® [17] have been independently analyzed by two software architects and ranked according to their quality. Quantitative software measures of these components have been compiled.

4.1. Data sets

EvIdent® is a user-friendly, algorithm-rich, graphical environment for the detection, investigation, and visualization of novelty/change in a set of images as they evolve in time or frequency. It is written in Java and C++ and is based upon VIStA, an application-programming interface (API) developed at the National Research Council. While Java is a platform-independent programming language that increases programmer productivity [15], it is computationally inefficient with numerical manipulations of large multidimensional datasets. As the result, all computationally intensive algorithms are implemented in C++.

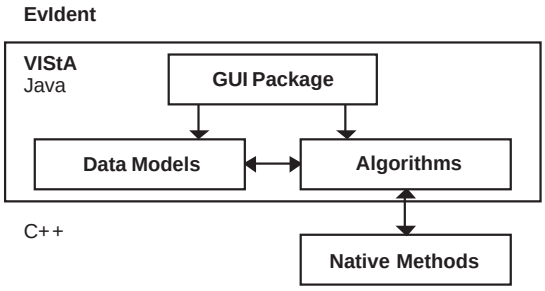

Fig. 11. EvIdent® software model with the VIStA API.

The VIStA API is written in Java and offers a generalized data model, an extensible algorithm framework, and a suite of graphical user interface constructs. Using the Java Native Interface, VIStA allows seamless integration of native C++ algorithms. Fig. 11 shows the class model and relationships between the three main Java packages used in VIStA.

All Java-based EvIdent®/VIStA objects have been used in this study (C++-based objects were excluded). For each of the 312 objects, two system architects, $A1$ and $A2$, were asked to independently rank its quality (low, medium, or high) with respect to extensibility, maintainability, efficiency, and clarity. The rankings were based entirely on the subjective assessments done by the architects. If an object was assessed to be in need of significant rewriting, then it received the label, *low*. If an object was deemed to be so well written that it would have been included in our coding standards document, as an example of good programming practice, then the object received the label, *high*. A label of *medium* was assigned otherwise.

A set of $n = 19$ software metrics were compiled for each object. Table 2 lists the minimum, maximum, and mean for each object. *NM*, *IC*, *KIDS*, *SIBS*, and *FACE* are the respective number of methods, inner classes, children, siblings, and implemented interfaces for each object. *DI* is the object's depth of inheritance (including interfaces). The features *rCC* and *rCR* are the respective ratios of lines of code to lines of comments and overloaded inherited methods to those that are not. *CBO* is a measure of coupling between objects, a count of the number of other objects to which the corresponding object is coupled.

RFO is a measure of the response set for the object, that is, the number of methods that can be executed in response to a message being received by the object. Note that the standard Java hierarchy was not included in the calculations of *IC*, *KIDS*, *SIBS*, *FACE*, *DI*, and *RFO*.

LCOM, the lack of cohesion in methods, is a measure that reveals the disparate nature of an object's methods. If I_j is the set of instance variables used by method j, and the sets P and Q are defined as

$$P = \{(I_i, I_j) \,|\, I_i \cap I_j = \emptyset\},$$
$$Q = \{(I_i, I_j) \,|\, I_i \cap I_j \neq \emptyset\}$$

then, if $\text{card}(P) > \text{card}(Q)$, $LCOM = \text{card}(P) - \text{card}(Q)$, otherwise $LCOM = 0$ [16].

LOC, *TOK*, and *DEC* are the number of lines of code, tokens, and decisions, respectively. The final feature, *WND*, is a weighted measure of the number of nested decisions (the greater the nesting the greater the associated weight). There are also *mLOC*, *mTOK*, *mDEC*, and *mWND* representing mean values per method of *LOC*, *TOK*, *DEC*, and *WND*.

Table 2
Set of software measures

Feature	Min	Max	Mean
NM	0	162	14.4
LOC	0	5669	296.7
mLOC	0	108	15.9
TOK	0	26,611	1387.2
mTOK	0	16,383	1051.2
DEC	0	582	25.1
mDEC	0	21	1.6
WND	0	1061	51.6
mWND	0	91	3.4
IC	0	20	0.7
DI	0	3	0.4
KIDS	0	10	0.3
SIBS	0	16	1.0
FACE	0	5	0.5
rCC	0	348	11.3
rCR	0	2.5	0.1
CBO	0	11	1.0
LCOM	0	9018	193.0
RFO	0	1197	66.1

An attribute, Quality Assessment (QA), has been identified as output attribute of an object. QA represents ranking of the quality assessment performed by the architects where 1 means *low*, 2 means *medium* and 3 means *high*.

4.2. Analysis of results

Synthesis of models representing the architects is performed using the GA-based approach. Software measures representing objects are inputs to the models. Granulation of input is done by defining 3 fuzzy sets for each software measure with the following linguistic labels: *small*, *medium* and *large*. The output of the models represents quality of the objects. Discrete character of the output (1—low quality, 2—medium quality, 3—high quality) has lead to some modifications of a fitness function being used in the experiments. An element representing a number of properly evaluated objects has become a part of a fitness function.

The models are developed for each architect independently. 2/3 of the data are used as a training set during construction of the models, and the remaining 1/3 is used as a test set. Applied Genetic Algorithm has had a tournament selection, one-point crossover, a uniform mutation, and a population of size 200, crossover probability of 0.85 and mutation probability of 0.45. The algorithm has used elitist strategy and run for 500 generations.

A series of experiments for two parameters controlling the structure of the network, the level of connectivity and the number of AND nodes in the hidden layer, has been carried out. The training and test set results for the first architect are presented in Table 3 where the performance index is the average error per data point. These quantities present the results independently from the size of

Table 3
The results of the GA optimization of the network for the first architect using the training (a) and test (b) sets (3 membership functions, the case in boldface is used in further optimization of the model)

	Connectivity no.			
Hidden nodes no.	2	3	4	5
(a) *Training data set—after GA, before tuning*				
2	0.2347	0.2379	0.2505	0.2550
3	0.2467	0.2495	0.2860	0.2993
4	0.2557	0.2573	0.2944	0.3195
5	0.2420	0.2484	0.2670	0.2803
6	0.2488	0.2522	0.2707	0.2759
7	0.2469	0.2491	0.2423	0.2893
8	0.2448	0.2464	0.2663	0.2772
9	0.2482	0.2317	0.2524	0.2685
10	**0.2250**	0.2397	0.2617	0.2739
(b) *Testing data set—after GA, before tuning*				
2	0.2471	0.2506	0.2643	0.2577
3	0.2521	0.2610	0.2851	0.2992
4	0.2612	0.2701	0.2953	0.3178
5	0.2468	0.2611	0.2674	0.2901
6	0.2608	0.2562	0.2718	0.2824
7	0.2538	0.2573	0.2559	0.2914
8	0.2559	0.2516	0.2661	0.2868
9	0.2584	0.2455	0.2559	0.2679
10	**0.2414**	0.2489	0.2680	0.2760

the training and testing sets. The table shows the values of the performance index of networks with different number of hidden nodes (rows) and different connectivity number (columns).

It can be said that for a fixed connectivity, the increase of the size of the hidden layer results (in general) in lower values of the performance index. A number of hidden nodes represents a number of if–then rules which can be extracted from the network. It means that an increased number of rules leads to better "coverage" of the input data space. At the same time the increased number of hidden nodes (if–then rules) leads to difficulties with comprehension of the results. Therefore, we have limited the number of the hidden nodes to ten.

It is also interesting to learn that the increased connectivity for a fixed size of the hidden layer gives rise to higher values of the performance index. The experiments have been designed in a way that a given connectivity is "forced" during a given experiment. It means that if the connectivity is three then three inputs have to be attached to a single hidden node. It seems that the nature of the data used is such that too many number of different attributes in the "if" part of the rule does not lead to improvement of the performance index.

The "meaningful" connections of the AND neurons are those equal to zero. For the OR neurons, the connections equal to 1 are essential. The architecture of the constructed network is presented in Fig. 12.

In the second development phase a blueprint of the network, produced by GA, is an object of parametric optimization. The gradient-based learning starts with connections established by GA optimization. At the beginning the weights of the connections from input nodes to hidden nodes

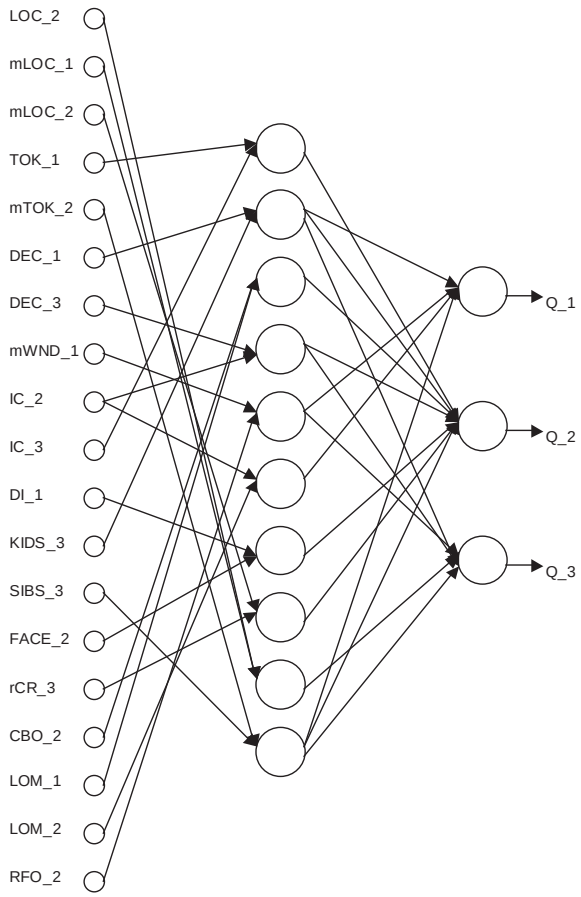

Fig. 12. A network generated by the GA optimization process "representing" the first architect.

are equal to zero, and the weights of connections from the hidden notes to the output nodes are one. The gradient-based learning process modifies these weights. This enhancement aims at further reduction in the value of performance index, Table 4.

The process of modification of the weights can results in a modification of the structure of the network. If the weights of connections to AND nodes increase above 0.5, and if the weights of connections to OR node decrease below 0.5 then such connections have very little influence on the results. In such case, these connections can be removed from the network. A new architecture of the network representing only valid connections (with weights below 0.5 for AND nodes, and above 0.5 for OR nodes) is presented in Fig. 13.

The dominant (the weights <0.2 for AND, and >0.8 for OR) *if-conditions* for the network representing the first architect are:

if
NUMBER_OF_INNER_CLASSES is *medium* (IC_2)
and
LACK_OF_COHESION is *medium* (LOM_2)

Table 4
The results of the parametric optimization of the network for the first architect for the training (a) and testing (b) sets

	Connectivity no.			
Hidden nodes no.	2	3	4	5
(a) *Training data set—after tuning*				
2	0.1902	0.1909	0.1887	0.1914
3	0.1900	0.1892	0.1911	0.1916
4	0.1907	0.1835	0.1875	0.1909
5	0.1856	0.1873	0.1884	0.1934
6	0.1877	0.1865	0.1902	0.1892
7	0.1868	0.1861	0.1853	0.1870
8	0.1860	0.1839	0.1852	0.1889
9	0.1880	0.1842	0.1879	0.1864
10	**0.1831**	0.1852	0.1843	0.1856
(b) *Testing data set—after tuning*				
2	0.2047	0.2071	0.2054	0.2020
3	0.1998	0.2002	0.2069	0.2045
4	0.2021	0.1990	0.1994	0.2051
5	0.1978	0.1995	0.2018	0.2093
6	0.2042	0.1982	0.2042	0.2052
7	0.2018	0.2018	0.2030	0.1939
8	0.1981	0.1994	0.1974	0.2020
9	0.2004	0.1992	0.2038	0.1990
10	**0.1987**	0.1957	0.2016	0.1964

then
ESTIMATED_QUALITY is *small* (Q_1)
if
NUMBER_OF_TOKENS is *small* (TOK_2)
and
NUMBER_OF_INNER_CLASSES is *medium* (IC_2)
or
MEAN_NUMBER_OF_TOKENS is *medium* (mTOK_2)
and
NUMBER_OF_SIBLINGS is *large* (SIBS_3)
then
ESTIMATED_QUALITY is *medium* (Q_2)
if
NUMBER_OF_DECISIONS is *small* (DEC_1)
and
NUMBER_OF_CHILDREN is *large* (KID_3)
then
ESTIMATED_QUALITY is *large* (Q_3)

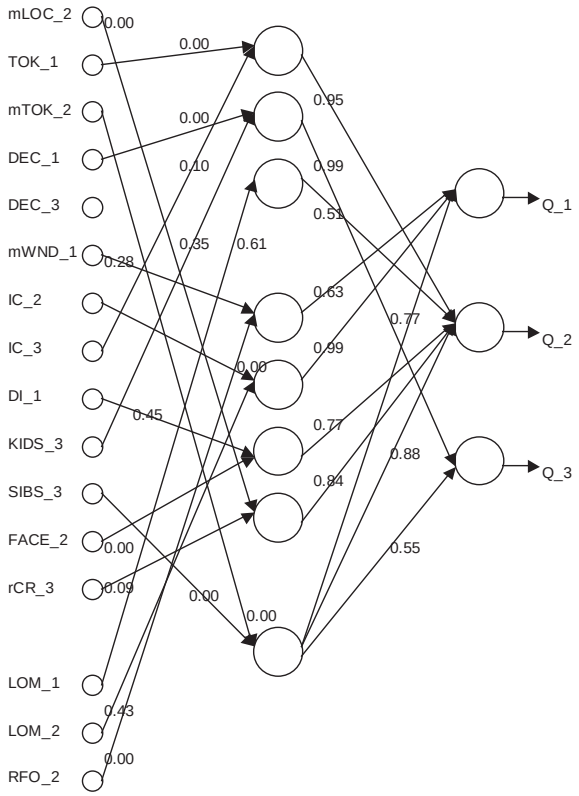

Fig. 13. A network generated by the GA optimization process and tuned by parametric optimization "representing" the first architect; only valid connections are shown.

The same set of experiments has been repeated using data generated by the second architect. In this case, the resulting network is presented in Fig. 14. The dominant *if-conditions* are the following:

if
MEAN_NUMBER_OF_DECISIONS is *small* (mDEC_1)
and
NUMBER_OF_IMPELMENTED_INTERFACES is *large* (FACE_3)
or
NUMBER_OF_DECISIONS is *small* (DEC_1)
and
MAEN_ NUMBER_OF_NESTED_DECISIONS is *medium* (mWND_2)
then
ESTIMATED_QUALITY is *small* (Q_1)
if
NUMBER_OF_INNER_CLASSES is *large* (IC_3)
and
NUMBER_OF_IMPELMENTED_INTERFACES is *medium* (FACE_2)

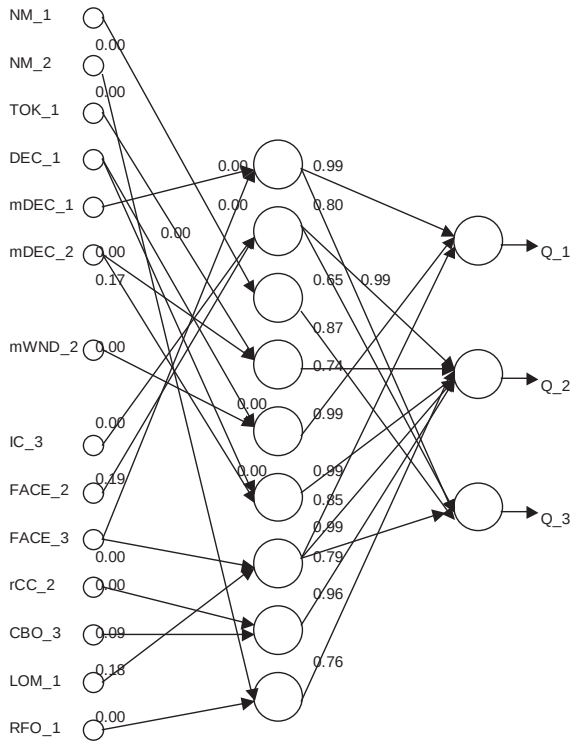

Fig. 14. A network generated by the GA optimization process and tuned by parametric optimization "representing" the second architect; only valid connections are shown.

or
NUMBER_OF_DECISIONS is *small* (DEC_1)
and
MEAN_NUMBER_OF_DECISIONS is *medium* (mDEC_2)
or
NUMBER_OF_IMPELMENTED_INTERFACES is *large* (FACE_3)
and
LACK_COHESION_METHODS is *small* (LOM_1)
then
ESTIMATED_QUALITY is *medium* (Q_2)
if
NUMBER_OF_METHODS is *small* (NM_1)
then
ESTIMATED_QUALITY is *large* (Q_3)

5. Module development efforts

By its definition, the Experience Factory embraces all kinds of knowledge related to software process development. Development of software is not only design and construction but also testing

Table 5
List of attributes of the MIS data

LOC	Length of code with comments
CL	Length of code without comments
TCHAR	Number of characters
TCOMM	Number of comments
MCHAR	Number of comment characters
DCHAR	Number of code characters
N	Program length, $N = N_1 + N_2$, where N_1 is the total number of operators, and N_2 is the total number of operands
NE	Estimated program length, $NE = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$ where η_1 is the number of unique operators and η_2 is the number of unique operands
NF	Jensen's program length, $NF = (\log_2 \eta_1)! + (\log_2 \eta_2)!$
VG	McCabe's cyclomatic number, is one more than the number of decision nodes in the control flow graph
BW	Belady's bandwidth metric, where $BW = 1/nSUM(iL_i)$ and L_i represents the number of nodes at level i in a nested control flow graph of n nodes, this metric is indicative of the average level of nesting or width of the control flow graph representation of the program
NC	Number of changes

and different forms of maintenance such as corrective and adaptive ones. In this case, it is important to know the relationships between the construction of objects and the number of defects and modifications that objects have to go through. A model expressing relationships between software measures of software components and the number of needed modifications of each component needs to be constructed. Knowledge about these relationships can lead to increased accuracy of estimates of software development processes, mostly during the phases of testing and maintenance.

5.1. Data set—a brief description

The Medical Imaging System (MIS) is a commercial software system consisting of approximately 4500 routines. It is written in Pascal, FORTRAN and PL/M assembly code. In total, there are about 400,000 lines of code. MIS development took 5 years and the system has been in commercial use at several hundred sites.

The MIS data set [10] contains a description of 390 software modules that comprise the system. The description is done in terms of 11 software measures (size of the code, McCabe complexity measure, Jensen complexity measure, etc.) and the number of changes made to each module due to faults discovered during system testing and maintenance. A list of all attributes (software measures) is presented in Table 5, and their min, max and mean values in Table 6.

Table 6
Set of software measures

Feature	Min	Max	Mean
LOC	0	5669	296.7
CL	0	108	15.9
TCHAR	0	26,611	1387.2
TCOMM	0	16,383	1051.2
MCHAR	0	582	25.1
DCHAR	0	21	1.6
N	0	1061	51.6
NE	0	91	3.4
NF	0	20	0.7
VG	0	3	0.4
BW	0	10	0.3
NC	0	16	1.0

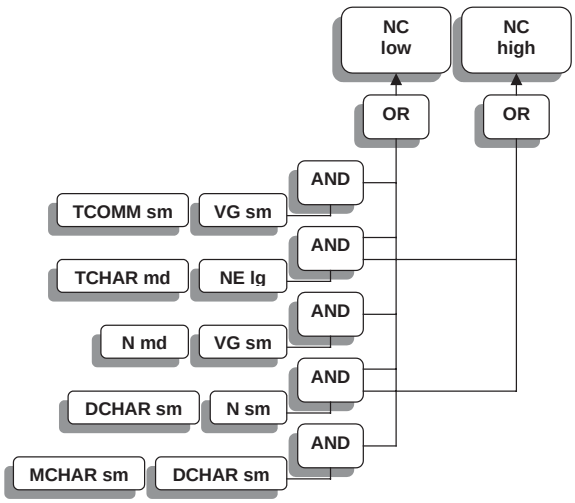

Fig. 15. An optimal structure of a two-output fuzzy network for the MIS data as the result of GP optimization.

5.2. Analysis of results

The fuzzy models are developed using 2/3 of the data points from the MIS data with the rest used as a test set. The number of fuzzy sets is 3 in each input space (attribute), and 2 in the output space. All fuzzy sets are described by means of Gaussian membership functions.

The GP used in the experiments implements tournament selection, elitist strategy, and two kinds of crossover operation and two kinds of mutation operation that have been described in Section 3.4. Probabilities of crossover and mutation have been set to 0.9 and 0.3 respectively. The experiments have been run with a population size of 200, and a number of generations equal to 1000.

The experiment is concerned with the development of a model with two outputs. Each of the outputs represents a single fuzzy set defined in the output space. The optimized structure of such model, the result of GP optimization, is shown in Fig. 15. Fig. 16 illustrates the course of the

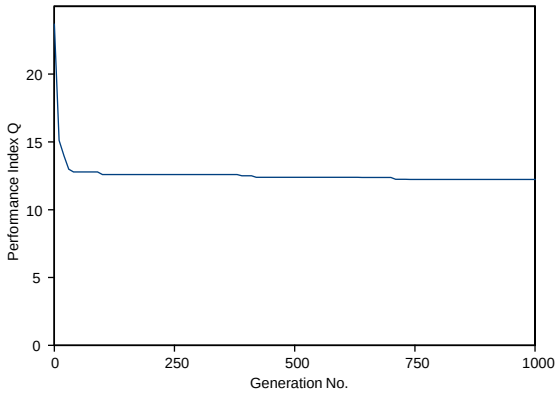

Fig. 16. Performance index Q in successive generations of GP optimization.

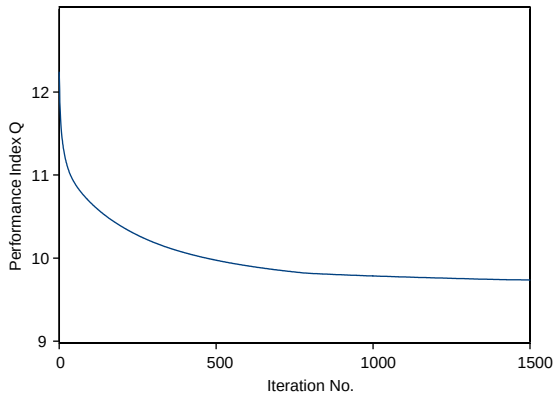

Fig. 17. Performance index Q in successive learning epochs for parametric learning of fuzzy neural network representing the MIS data.

optimization process by changes in the values of the performance index Q. The normalized performance index $\tilde{Q}$ (that is the value of Q divided by the number of data points) of the optimal structure is equal to 0.0235. Using the test set, $\tilde{Q}$ rises slightly to 0.0301. The optimal structure of the model contains 7 out of 11 attributes of the MIS data.

The fuzzy model obtained in the process of GP optimization is then optimized parametrically through a gradient-based learning. The learning rate (α) is set to 0.005 and it runs for 1500 learning epochs. The values of the performance index versus successive learning epochs are shown in Fig. 17. The normalized performance index, $\tilde{Q}$, of the optimal fuzzy neural network, after structural and parametric optimization, is equal to 0.0187, and for the test set is 0.0269. The value of the index, compared with its initial value after the structural optimization, improves by 20% in the case of training data, and by 11% in the case of testing data.

The structure of the MIS data model after parametric optimization is presented in Fig. 18. In this model, all connections have been augmented by the values, which change the significance of some input sets and the rules. The dominant *if-condition* for membership function *low* is "**N** is *medium*

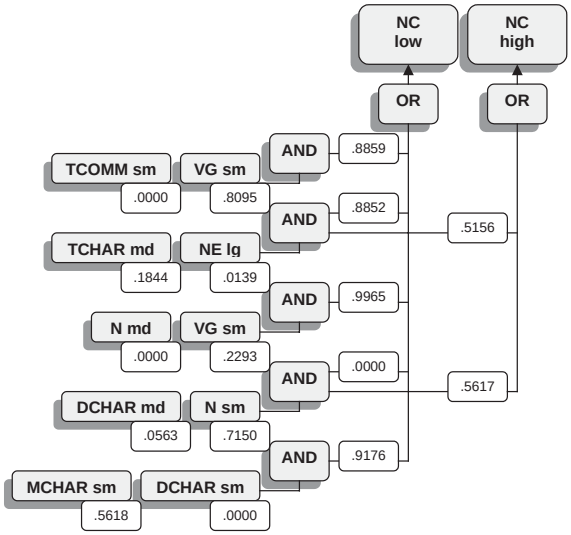

Fig. 18. Structure of a two-output fuzzy network after parametric optimization representing the MIS data.

AND **VG** is *small*". In the case of *high* membership function, there are only two *if-conditions* "**TCHAR** is *medium* AND **NE** is *large*" and "**DCHAR** is *medium* AND **N** is *small*" and both of them have similar significance represented by the value around 0.5.

6. Conclusions

In this study, we have proposed a general design methodology for granular-based models fulfilling two fundamental requirements of granular modeling: accuracy and transparency. The optimization tandem of evolutionary computing and gradient-based learning of fuzzy neural networks naturally supports structural and parametric optimization of the models that helps achieve accuracy. The transparency of the model is accomplished by subscribing to the logic-oriented architecture of the fuzzy neural networks.

The proposed approach has been used to build elements of the Experience Factory, which supports software development projects by supplying experience obtained from previous software projects.

A genetic algorithm-based method has been applied to construct a process model of quality assessment of software objects. The model has led to the recognition of significant relationships between software measures, which are essential for evaluation purposes. Software managers can use such a model to support the quality assessment process. Developers can employ these models for estimation of results of their work. Based on work done so far, developers can semi-automatically synthesize models, modify them to their experiences, and use these models in a preliminary process of quality evaluation to identify poorly implemented objects.

The second proposed approach, using a genetic programming-based method, has been used to development of a model representing development efforts. Knowledge about relationships between software measures representing modules and the number of needed modifications of each module

is essential for software managers and developers for realistic estimations of work needed in the testing and maintenance phases of software development.

Acknowledgements

Support from the Natural Sciences and Engineering Research Council (NSERC) and the Alberta Software Engineering Research Consortium (ASERC) is gratefully acknowledged.

References

[1] T. Back, D.B. Fogel, Z. Michalewicz (Eds.), Evolutionary Computations I, Institute of Physics Publishing, Bristol, 2000.
[2] V.R. Basili, L.C. Briand, W. Melo, A validation of object-oriented design metrics as quality indicators, IEEE Trans. Software Eng. 22 (10) (1996) 751–761.
[3] V.R. Basili, G. Caldiera, H.D. Rombach, The Experience Factory, in: J. Marciniak (Ed.), Encyclopedia of Software Engineering, vol. 1, Wiley, NY, 1994, pp. 469–476.
[4] S.R. Chidamber, C.F. Kemerer, A metrics suite for object-oriented design, IEEE Trans. Software Eng. 20 (6) (1994) 476–493.
[5] D.B. Fogel, Evolutionary computation, Toward a New Philosophy of Machine Intelligence, IEEE Press, Piscataway, 1995.
[6] D.E. Goldberg, Genetic algorithms in search, optimization, and machine learning, Addison-Wesley, Reading, MA, 1989.
[7] J.H. Holland, Adaptation in natural and artificial systems, 2nd Edition, MIT Press, Cambridge, MA, 1992.
[8] P.G. Korning, Training neural networks by means of genetic algorithms working on very long chromosomes, Internat. J. Neural Systems 6 (3) (1995) 299–316.
[9] J.R. Koza, Genetic Programming, MIT Press, Cambridge, MA, 1992.
[10] J.C. Munson, T.M. Khoshgoftaar, Software metrics for reliability assessment, in: M.R. Lyu (Ed.), Software Reliability Engineering, Computer Society Press, Los Alamitos, CA, 1996, pp. 493–529.
[11] W. Pedrycz, Fuzzy Sets Engineering, CRC Press, Boca Raton, FL, 1995.
[12] W. Pedrycz, Computational Intelligence: An Introduction, CRC Press, Boca Raton, FL, 1997.
[13] W. Pedrycz, Granular computing: an introduction, in: M.H. Smith, W.A. Gruver, L. Hall (Eds.), Proc. Joint 9th IFSA World Congr. and 20th NAFIPS Internat. Conf., IEEE Press, Vancouver, Canada, July 25–28, 2001, pp. 1349–1354.
[14] W. Pedrycz, F. Gomide, An Introduction to Fuzzy Sets, Analysis and Design, MIT Press, Cambridge, MA, 1998.
[15] J.F. Peters, W. Pedrycz, Software Engineering: An Engineering Approach, Wiley, New York, USA, 2000, pp. 505–551.
[16] G. Phipps, Comparing observed bug and productivity rates for Java and C^{++}, Software Practice and Experience 29 (1999) 345–358.
[17] N.J. Pizzi, R.A. Vivanco, R.L. Somorjai, EvIdent: a functional magnetic resonance image analysis system, Artificial Intelligence Med. 21 (2001) 263–269.
[18] M. Russo, Genetic fuzzy learning, IEEE Trans. Evolutionary Comput. 4 (3) (2000) 259–273.
[19] W.M. Spears, Crossover or mutation? in: L.D. Whitley (Ed.), Foundations of Genetic Algorithms Workshop, Morgan Kaufmann, San Mateo, CA, 1992, pp. 221–237.
[20] X. Yao, Evolving artificial neural networks, Proc. IEEE 87 (9) (1999) 1423–1447.