Diagnosis based on genetic fuzzy algorithms for LTE Self-Healing

E. J. Khatib[1], R. Barco[1], A. Gómez-Andrades[1], I. Serrano[2]

[1] Universidad de Málaga, Communications Engineering Dept., Málaga, Spain
[2] Ericsson, PBO RA Continuous Analysis

Self-Organizing Networks (SON) mechanisms reduce Operational Expenditure (OPEX) in cellular networks, whilst enhancing the offered quality of service. Within SON, self-healing aims to autonomously solve problems in the radio access network and to minimize their impact on the user. Self-healing comprises automatic fault detection, root cause analysis, fault compensation and recovery. This paper presents a root cause analysis system based on fuzzy logic. A genetic algorithm is proposed for learning the rule base. The proposed method is adapted to the way of reasoning of troubleshooting experts, which ease knowledge acquisition and system output interpretation. Results show that the obtained results are comparable or even better than those obtained when the troubleshooting experts define the rules, with the clear benefit of not requiring the experts to define the system. In addition, the system is robust, since fine tuning of its parameters is not mandatory.

***Index Terms*—Self-Organizing Networks, self-healing, troubleshooting, root cause analysis, genetic algorithms, supervised learning, fuzzy systems**

I. Introduction

In the last decade, cellular mobile networks have grown rapidly in size and complexity. Operation and maintenance of these networks is a difficult challenge because of the ever-growing number of users and a demand for reliability. For this reason, the NGMN Alliance [1] came up with the definition of *Self-Organizing Networks (SON)* as a set of principles and concepts to add automation to mobile networks so that they require less maintenance than traditional networks while improving service quality. Subsequently, 3GPP proposed a set of use cases and specific SON functionalities for next generation mobile communication networks, that is *Long Term Evolution (LTE)* [2], and identified SON as one of the key features of LTE-Advanced (LTE-A) [3]. Three main aspects conform the SON definition: *self-configuration*, *self-optimization* and *self-healing*. This study contributes to the research effort in **self-healing** [4] [5]. Self-healing encompasses the automatic functionality aimed to solve problems (traditionally called troubleshooting) as they emerge in the network. It is a substitute of manual troubleshooting, which currently is the only way network problems are being confronted. Automatic troubleshooting is a great competitive advantage since offloading the Operation and Maintenance (O&M) experts from the task of solving common problems, especially related to the access networks, greatly reduces Operational Expenditure (OPEX). The reduced response time of automatic troubleshooting also increases the quality of service.

Self-healing consists of fault detection, root cause analysis (diagnosis), compensation and fault recovery. Recently, some research effort has started to be carried out on these topics. However, so far there are no commercial products that implement a totally automated troubleshooting tool, although there is an emerging research effort into the topic. In [6] fault detection is based on monitoring the throughput. A sector is considered problematic if it falls under a certain threshold that is calculated according to the recorded average throughput. There is no diagnosis stage. The neighboring sectors are then modified to compensate the problematic sector. A neural network based approach is described in [7] to detect problems in 3G networks. The neural networks are trained to recognize the normal behavior.

In [8] a framework for diagnosis based on bayesian networks is proposed. A Knowledge Acquisition Tool for defining the parameters of the bayesian network is defined in [9]. The UniverSelf project [10] proposes to combine bayesian networks with case-based reasoning [11] [12].

The COMMUNE project [13] is targeted towards the study of uncertainty in mobile networks. It highlights uncertainty as the main cause that *self-x* (self-healing amongst others) functions are underdeveloped. In [14] it is determined that bayesian networks are a complex approximation for troubleshooting, given the difficulty to obtain expert knowledge as the required conditional probabilities. Thus, COMMUNE is inclined to case based reasoning, such as the algorithm described in [15], where a PI (Performance Indicator) vector is compared against a database of known cases, and the most similar is selected as the diagnosed cause. It has an implicit supervised learning capability, as experts add cases to the database. Some research effort has also been put into compensation. [16] describes the problem of cell outage detection and compensation. [17] and [18] describe a method of compensation based on modifying the antenna tilt.

Among self-healing functions diagnosis is especially critical, since operators spend long periods of time finding out the root causes of problems. Based on the previous references, it can be concluded that although some approaches have been proposed for diagnosis, they have not been accepted for implementation in real networks. The main cause of this failure is that these systems require a high collaboration degree from the troubleshooting experts, since the knowledge transfer forces them not only to dedicate time, but also the effort of learning the topics related to the selected technique. Even if the system is built from solved cases or from a knowledge

acquisition tool, operators feel reluctance to use the automatic diagnosis system because they do not understand its way of reasoning, which is far from the workflow normally used by troubleshooting experts.

In this paper, diagnosis system based on fuzzy logic [19] is used. Fuzzy logic models the process of human thinking, translating the system input values to sets (*fuzzy sets*) that are easy to understand and to link through heuristic rules (*fuzzy rules*). Therefore, since the expert knowledge is represented by rules, converting the flowcharts that troubleshooting experts normally use into the rules required by the fuzzy system and vice versa is straightforward. This is a large advantage compared to other solutions, both for defining the system and for the operators acceptance in real networks. Fuzzy logic has been applied to troubleshooting in other fields such as industrial processes and machinery, among others. In mobile networks, it has been used for optimization [20] [21], but not for self-healing.

Traditionally, the preferred method for creating the fuzzy rules has been by knowledge acquisition. But as the systems controlled by fuzzy logic become more complex, this task is harder and the help of automated methods is required. Since learning algorithms use heavy CPU and memory resources, the advent of faster and cheaper computers in the last decade is helping this trend. This paper proposes to use a genetic learning algorithm to obtain troubleshooting rules from solved troubleshooting cases. These rules will then be used on a fuzzy logic based troubleshooting system. The learning task will be done by observing the work of the troubleshooting experts through solved problems. This method requires less intervention from the experts compared to the methods previously proposed in the literature, that is either interviews with experts or requiring the experts to define the parameters related to the given artificial intelligence technique (fuzzy logic in this case). This will increase the expert collaboration degree, since the effort required from them is much smaller.

This paper is organized as follows. In Section II the basic concepts used in this paper are introduced. In Section III, the proposed genetic algorithm and the method for diagnosis in mobile networks are described. In Section IV, the proposed system is evaluated. Finally, the conclusions are discussed in Section V.

II. Problem formulation

Troubleshooting in mobile networks is a procedure that has traditionally been manual. It consists of three main tasks:

1) **Detection**: to identify the sectors with degraded performance.
2) **Diagnosis**: to determine the cause of the problem.
3) **Recovery**: to take actions to solve the problem (repair HW, configuration corrections, etc).

While the fault is identified and solved, since this may take some time, **compensation** may be triggered, that is neighboring cells may provide service to the users under the problematic cell.

Manual troubleshooting is usually a process of application of certain heuristic rules for determining a diagnosis given the observation of some symptoms. In mobile networks, the symptoms are observed as abnormal values of Performance Indicators (PIs).

Experts use a top-down approach for the detection stage; they monitor some high level Key PIs (KPIs, such as the number of dropped and blocked calls, failed handovers, etc) aggregated on the whole network (or on a group of sectors). When a degradation is detected in these high level KPIs, a list of the worst offenders is obtained identifying the sectors that are causing degradation. Once those sectors are identified, the diagnosis stage further screens additional PIs, active measurements and configuration audits. Once a possible cause has been determined out of the observations and the experience of the expert, a correction is applied, and depending on its outcome, the problem may be considered solved.

In this paper, fuzzy logic is used for automating the diagnosis stage of a troubleshooting system. A set of representative PIs is selected and fed to a fuzzy logic controller (FLC) [22] that contains the appropriate rules for determining the problems. Fuzzy sets and fuzzy rules are modeled to imitate the process of thinking of the troubleshooting experts. This is achieved in two possible ways:

- Knowledge acquisition: the rules are derived directly from the experts by interviewing them [9].
- Learning algorithms: the rules are derived by an algorithm from a training set. The training set consists of a list of specific cases that are either labeled (supervised learning) [23] or not labeled (unsupervised learning) [24]. In this study, only supervised learning is considered.

The aim of the learning algorithm proposed in Section III-C is to obtain the rules from a set of training pairs (vector of PIs and a label identifying the problem) fed by experts. This changes the focus of the experts in the process of knowledge acquisition. Instead of dealing with the parameters of the fuzzy system (which requires training them in the details of fuzzy logic controller design), they are only required to choose the most representative troubleshooting cases.

III. Fault management based on genetic algorithms

A. Fuzzy logic controllers

Fuzzy logic is a branch of artificial intelligence modeling human thinking [19]. To do this, it transforms numerical values of variables into descriptive values (such as *"high"* or *"low"*).

Several *fuzzy sets* (S_1, S_2, ...) are defined over the domain of a *crisp* variable. A crisp variable is a common numerical non-fuzzy variable, and its domain is the *universe of discourse* (U). A fuzzy set comprises the values of U that have a common characteristic as perceived by a human (for instance, *high* or *low* values for a PI). Each fuzzy set S_i has an associated *membership function* $\mu_{Si} \in [0, 1]$ that defines the degree of truth of each crisp value belonging to that fuzzy set. In Fig. 1 an example of how membership functions work is depicted. A crisp variable x has two fuzzy sets defined on its domain: S_1 and S_2. For a given crisp value x_1 of x, the membership degree for each set is given by $\mu_{S1}(x_1)$ and $\mu_{S2}(x_1)$, the membership functions of sets S_1 and S_2, respectively. A *fuzzy*

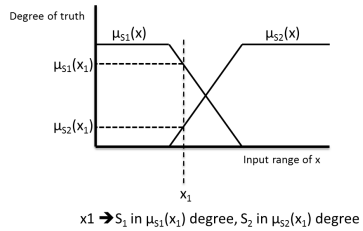

Fig. 1. Fuzzification of a crisp variable

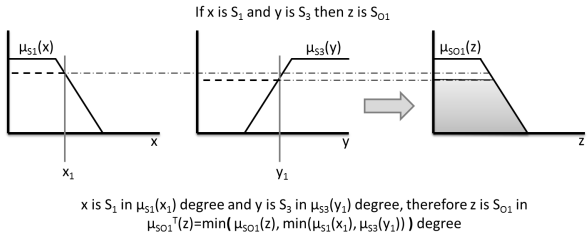

Fig. 2. Linguistic reasoning

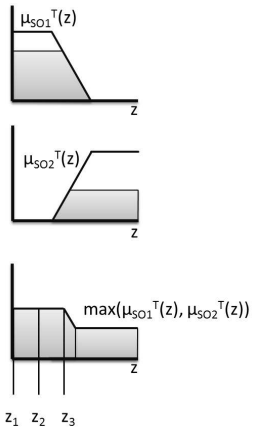

Fig. 3. Defuzzification process

variable is formed by the linguistic labels identifying a fuzzy set and their membership functions. The process of converting a crisp value to a fuzzy value is called *fuzzification*. In Fig. 1, the fuzzified value of x_1 is "S_1 in $\mu_{S1}(x_1)$ degree and S_2 in $\mu_{S2}(x_1)$ degree".

Fuzzy logic controllers (FLC) are algorithms that use the principles of fuzzy logic to assign values to output variables based on some input values. FLCs are often used in control systems where the input variables reflect the state of the system and the output variables are control actions. In this paper, a FLC is used for automatic diagnosis in mobile communications networks, the inputs being PIs, alarms and configuration parameters, and the outputs the root cause and possible solutions. A FLC applies three consecutive processes: *fuzzification*, a *fuzzy reasoning* that assigns fuzzy values to output variables based on fuzzy values of input variables and *defuzzification*, that transforms the fuzzy value of the output variable into a crisp value.

Fuzzy reasoning in a FLC is done through *If ... Then ...* rules. These *fuzzy rules* are composed of two main parts: the *antecedent* (the *if* part) and the *consequent* (the *then* part). The degree of truth of the consequent is obtained by calculating the degree of truth of the antecedent.

The antecedent contains assertions about input variables belonging to fuzzy sets (for example "x_1 is S_1"). The degree of truth of these assertions is the degree of membership of the variables ($\mu_{S1}(x_1)$). Several assertions can be done in the same antecedent, joined by *AND* or *OR* operators. Usually in these cases, the degree of truth of the antecedent is the minimum of all the individual assertions (with *AND* operators) or the maximum (with *OR* operators).

The consequent contains an assertion about an output variable. The degree of truth of the antecedent modifies the membership function of the fuzzy set of the value assigned in the assertion, either by truncating it or by obtaining the product. The full process of assigning a degree of truth to a variable in the consequent based on the degree of truth of the antecedent is depicted in Fig. 2. The antecedent of the rule has two assertions: "x is S_1" and "y is S_3". The minimum degree of truth of both assertions (in this case, $\mu_{S3}(y_1)$) is assigned to the consequent. The degree of truth of the consequent truncates the membership function $\mu_{SO1}(z)$ of the fuzzy set S_{O1} assigned by the assertion "z is S_{O1}".

The crisp value of an output variable z can be inferred through the aggregation of the outputs of individual rules. A truncated membership function $\mu^{(T)}_{SO1}(z)$, $\mu^{(T)}_{SO2}(z)$, ... is obtained for each rule on the domain of the output variable according to the results of the linguistic reasoning. For each point of the domain of the output variable, the maximum degree of truth among the output membership functions is taken, that is a combined function $\mu_O(z)$ is defined as $\mu_O(z) = max(\mu^{(T)}_{SOi}(z))$. A crisp value is then obtained from this function in the *defuzzification* process. The crisp value can be taken according to a specific policy:

- *SOM* (Smallest Of Maximum): The smallest point with the maximum degree of truth.
- *Centroid*: The average of all points with the maximum degree of truth.
- *LOM* (Largest Of Maximum): The largest point with the maximum degree of truth.

Fig. 3 depicts this process. The output of two individual rules (functions $\mu^{(T)}_{SO1}(z)$ and $\mu^{(T)}_{SO2}(z)$) are aggregated, creating a combined membership function. The crisp value for variable z is one of the points where this new function is maximum. z_1, z_2 and z_3 are the values for z if the defuzzification method is SOM, centroid and LOM, respectively.

B. Genetic algorithms

This paper proposes a genetic algorithm to automatically learn the rules of the FLC used for diagnosis. Genetic algorithms [25] [26] imitate the process of natural selection, that is, they seek the solution of problems by a trial and error method, repeated generation after generation. All genetic algorithms have three common elements:

- A **population of individuals**, each being a possible solution to the problem. Each individual is a collection

Fig. 4. Format of training data

(*genome*) of traits (*gens*) that determine the behavior of the solution and are subject to variation when the individual reproduces.

- **Operators** that define the birth of new individuals. Specifically, there are two big groups of operators:
  - *Crossing*: when the traits of two individuals are combined to produce a new improved individual.
  - *Mutation*: when a new individual is created by copying another individual and introducing minor (usually random) changes.
- A **fitness function** that assigns a score to each solution based on its quality.

A typical genetic algorithm has three processes:

- **Reproduction**: new individuals are created either by crossing two chosen rules, by mutation, or a combination of both. Usually, a high mutation rate produces a high exploration ratio, since random changes are introduced. In some algorithms, the parent rules are eliminated once the reproduction is done.
- **Evaluation**: the fitness of each individual is obtained using the fitness function. This value will determine the probability of the individual to survive and reproduce in future generations.
- **Selection**: This process decides which rules survive to pass on to the next generation. The rules with a higher fitness have higher chances of surviving.

C. Genetic rule learning for diagnosis in cellular networks

In this section the genetic algorithm proposed for learning troubleshooting fuzzy rules from previously diagnosed examples (hereafter called cases) is described.

The training data is a set of cases following the format described in Fig. 4. A case is a vector consisting of crisp values for several PIs, and a class label corresponding to the diagnosed fault cause.

The statistical behaviour of each PI may be influenced by diverse aspects, including (but not limited to) the location of the originating sector, the distribution of the users or the configuration. Also, the requirements for each sector may vary, so what is considered a normal value for a certain sector, may be an abnormal value for another sector. Since the data vectors are fuzzified before being used for diagnosis or learning, this variability is managed in the fuzzification process. The fuzzy sets may vary from sector to sector, adapting to what is the meaning of the PI values for each sector. These fuzzy sets may also be parameterized to adapt to variables such as the time of the day to reflect the time-dependent behaviour of the PIs. The output of the fuzzification process is therefore a normalized version of the input vectors, that may then be used by the FLC or the learning process regardless of the original numerical value.

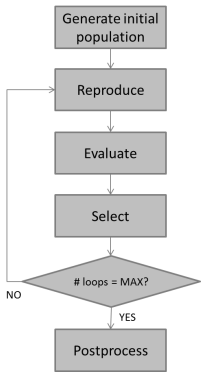

Fig. 5. Flowchart of the algorithm

In cellular networks, there are thousands of counters and indicators which can be used as symptoms for diagnosis. However, typically in the process of troubleshooting experts concentrate only on a limited subset of PIs to identify the root causes. In addition, the analyzed PIs may be different depending on the fault cause under study. Due to this fact, a fuzzy system designed for diagnosis in cellular networks should allow incomplete rules [27]. Incomplete fuzzy rules are rules that do not require a value for every input variable. The FLC can detect cases with more than a fault cause simultaneously only if incomplete rules are included. Therefore troubleshooting rules must only contain information about the PIs that are affected by a certain fault cause.

In case a new problem that was not present in the training set is fed to the FLC, the diagnosis will be wrong, since there is no information in the rules. It may either be a different diagnosis, or diagnosed as not being a problem. To prevent this situation, it is necessary to execute the learning algorithm whenever a new class of problems is found, either by manual inspection of the problems or by observing an unusual increase in diagnosis errors of the FLC.

The proposed algorithm, which includes the elements and processes described in Section III-B, is shown in Fig. 5. The features of the algorithm have been devised as follows:

- **Individuals**: each individual is a rule. The genome of an individual has two components:
  - **Antecedent** (the "*if* ..." part): a vector, $\{a_1, a_2, ..., a_L\}$, of length L equal to the number of inputs of the diagnosis system. Each entry a_i in this vector represents the fuzzy value that the PI k_i should take for the rule to be activated. If k_i should be low, then $a_i = 1$; if it should be high, then $a_i = 2$, and if the rule does not require any particular value for k_i, then $a_i = 0$. The last value of a_i allows the existence of incomplete rules. Incomplete rules are calculated at the end of the algorithm, and are used in the process of reproduction, but there are no individuals representing incomplete rules during the execution of the algorithm.
  - **Consequent**: (the "*then* ..." part): an integer, c, representing the problem diagnosed by the rule for the fuzzy values of the antecedent.

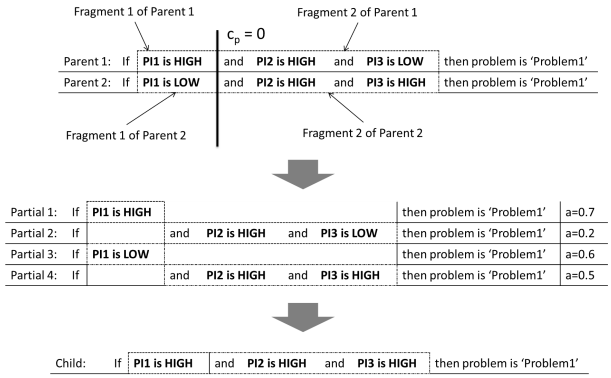

Fig. 6. Example of crossing two rules to obtain a child.

Each individual has an associated score W that represents its fitness. The initial population is generated with random valid values in the antecedent vector and the consequent.

- **Reproduction**: the rules are chosen randomly to reproduce. Crossing and mutation are done separately. For each individual, the probability of being chosen for reproduction by crossing is given by $P_{cross} = W \cdot R$, where W is the score of the rule, and R is the reproduction ratio. Only the best rules are chosen for crossing, so the best traits are passed on. For mutation, the probability is $P_{mut} = 1 - P_{cross}$, so the worst rules (that have not been killed) are given a chance to improve.
  Crossing (Fig. 6) happens between parent rules that have the same consequent. A random cutting point $c_p \in [0, L-1]$ is chosen. Each parent rule antecedent is divided into two fragments $[a_0 ... a_{c_p}]$ (first fragment) and $[a_{c_p+1} ... a_L]$ (second fragment). A score is given to each of the four fragments (two per parent). For this purpose, each fragment is extended with zeros into a valid antecedent of its own, and a partial rule is created with this antecedent. The partial rules are then evaluated with the training cases to see which of them are activated more often by calculating the average *degree of activation* ($\overline{DoA}$). The DoA of a rule for an input vector is the degree of truth of the antecedent. An *attractiveness* (a) value is assigned to each fragment:

$$a = \overline{DoA}_{fragment} \cdot W_{parent} \tag{1}$$

  where W_{parent} is the score of the parent rule. The child is then created with an antecedent that is the combination of the best (most attractive) first fragment with the best second fragment. The parents and the child are kept in the population.
  The *mutation* process creates several copies of an individual, each with one gene randomly altered. Either one of the components of the antecedent vector or the consequent are given a random value between the valid values it can take.
- **Evaluation**: the *score* W of the rules is calculated as the product of two separate terms:

$$W = B \cdot S_R \tag{2}$$

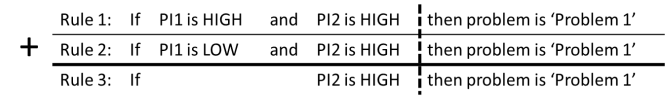

Fig. 7. Example of fusion of two rules.

The *base* (B) represents the statistic relevance of the rule and the *success rate* (S_R) represents the accuracy of the rule, that is the percentage of cases fulfilling the rule. The base is defined as:

$$B = 1 - \frac{1}{1 + \alpha \frac{c}{N}} \tag{3}$$

Where α is a parameter that adjusts the sensitivity of the base to uncommon cases (the higher is α, the easier it is to obtain a high base even with a low number of covered cases), c is the number of covered cases and N is the total number of cases in the training set. A case is considered covered when the DoA of the antecedent for that case is greater than a predefined threshold *Minimum Degree of Activation* (MDA).

- **Selection**: the algorithm finds the rules that have a score equal to 0 and eliminates them.
- **Postprocessing**: we propose to add an extra stage to the standard genetic scheme, where the final population is processed to create a rule set containing *incomplete rules*. Rules are grouped according to their consequents, and they are fused into incomplete rules. Two rules are fused if they are both correct. A rule is considered to be correct if its score is higher than the *Minimum Inclusion Degree* (MID). In that case, the contradictions between them are resolved by ignoring the non-common parts of the antecedent. Rules whose score is lower than MID are removed. An example of rule fusion is shown in Fig. 7. Since Rule 1 and Rule 2 have different restrictions over PI1, and if both rules are correct, then it will be considered that the value of PI1 is not relevant for detecting the cause 'Problem 1'. PI2 has the same restrictions in both rules, so the observation of a HIGH value in this PI leads to the detection of 'Problem 1'.

D. Parameters

The algorithm has 7 parameters:

- *Uncommon case sensitivity* (α) $\in (0, +\infty)$: adjusts the sensitivity of the algorithm to uncommon cases. A too low value may disregard cases that are important although they are uncommon. A too high value may give relevance to cases that are wrongly diagnosed.
- *Minimum inclusion degree* (MID) $\in (0, 1)$: minimum score a rule must have to be part of the final rule set.
- *Minimum degree of activation* (MDA) $\in (0, 1)$: indicates the minimum DoA of a rule on a case to consider that it covers it.
- *Initial rules* $\in [1, +\infty)$: Number of random initial rules. If none of the initial rules survives the first iteration of the algorithm, an *extinction* occurs and the algorithm stops.

- *Number of loops* $\in [1, +\infty)$: Number of iterations of the genetic algorithm. The number of loops must be high enough for the algorithm to converge to a correct solution.
- *Reproduction ratio (R)* $\in (0, 1]$: the proportion of cases that will reproduce (supposing all the cases have $W = 1$). This parameter is a population growth control.
- *Mutation multiply factor (MMF)* $\in [1, +\infty)$: Number of children that a rule produces when mutation takes place. A small value will produce a low exploration rate, whereas a too high value will increase the execution time.

IV. Evaluation

A. Case study

In cellular networks, the O&M can provide historical records containing the value of PIs and configuration parameters. However, after troubleshooting experts perform a root cause analysis of problematic cells, the diagnosed cause is normally not saved together with the value of the PIs. Therefore, there are no existing databases of classified fault cases in which to test new diagnosis systems, especially in LTE, which is starting to be deployed in current networks.

Due to these reasons, and in order to be as close as possible to the behavior of real networks (consequently, discarding using a simulator), a network emulator was built based on the knowledge of troubleshooting experts. The network emulator is simply a case generator, i.e. it produces fault causes and their associated PIs (labeled cases). The advantage of using the network emulator is that it can provide as many cases as needed, still being close to the real network behavior. In order to define the LTE network emulator, the following methodology was followed. Firstly, troubleshooting experts defined the most common categories of fault causes and related PIs for LTE. Secondly, the frequency of occurrence of each fault cause was defined (prior probabilities). Finally, the probability density function (PDF) of each PI conditioned to each fault cause was modeled either as a normal or a beta probability density function, which has been demonstrated to be an adequate distribution in cellular networks [8]. The parameters of the distributions were defined by the experts. The network emulator works by generating new cases according to the defined prior and conditional probabilities. The network emulator has been used to generate both training cases and testing cases.

The defined fault categories, each covering a broader group of causes, are the following:

- **SW problem**: There is a problem with the software in the site. Performance is degraded, but radio conditions are normal.
- **Coverage**: The sector has problems reaching some regions in its covering area.
- **Quality**: The uplink or downlink signal in the sector suffers interference.
- **Mobility**: Problem with handovers to/from neighboring sectors.

In addition, a "normal" category is included, for the cases where there are no problems in the network.

TABLE I
Prior probabilities

Fault category	Proportion (%)
SW Problem	13
Coverage	25
Quality	34
Mobility	28

TABLE II
PI modeling. PDF of each PI conditioned to an existing problem

PI	Type		Parameters/Cause				
			Nor	SW	Cov	Qual	Mob
Acc.	beta	α	2	12	1.391	450.3	2
		β	0.1	3	0.028	23. 7	0.5
Ret.	beta	α	17	11.756	10	11	9
		β	0.5	1.306	1.5	1.9	2
HOSR	beta	α	4	4.62	3	5	42.5
		β	0.02	0.024	0.02	0.04	7.5
RSRP	norm	avg	-70	-75	-107	-72	-80
		σ	3	6	5	7	10
RSRQ	norm	avg	-6.5	-6	-10	-13	-11
		σ	1.1	2	5	2	3

For this study, the probability of occurrence of each fault case is given in Table I. These probabilities are conditioned to the existence of a problem.

The most important PIs used to identify the previous fault causes are the following:

- **Accessibility**: it reflects the ability to establish a connection in the network. It is the inverse of the blocking rate. Its values range between 0 and 100%, and it is considered *normal* above 99% and *low* below 98%. These values are a commercial requirement.
- **Retainability**: it reflects the ability to end a call correctly. It is the inverse of the dropped call rate. Its values range between 0 and 100%, and it is considered *normal* above 99% and *low* below 98%. These values are also a commercial requirement.
- **Handover Success Rate (HOSR)**: percentage of initiated handovers that end successfully. Its values range between 0 and 100%. It is considered *normal* above 98.5% and *low* below 95%. These values are also a commercial requirement.
- **95 percentile of RSRP**: value of RSRP under which 95 percent of the samples fall. Experts consider it *low* if it is below -100 dBm and *high* if it is above -80 dBm.
- **95 percentile of RSRQ**: value of RSRQ under which 95 percent of the samples fall. Experts consider it *low* if it is below -23 dB and *high* if it is above -12 dBm.

The probability density functions of the PIs conditioned to the causes, together with their parameters are shown in Table II. As an example, Fig. 8 shows the PDF of the 95 percentile of RSRP conditioned to each fault cause.

B. Experimental design

With the emulator described in Section IV-A, 2000 training cases (containing 600 problems and 1400 normal cases) and

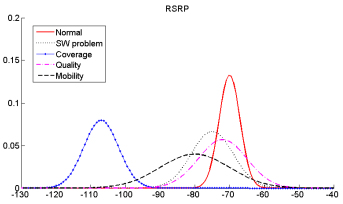

Fig. 8. PDF of the 95 percentile of RSRP PI.

5000 testing cases (1500 problems and 3500 normal) have been generated. Both sets contain the proportions for each problem defined in Table I. The algorithm proposed in this paper is first trained with the training cases, and afterwards, it is tested with the testing cases. This process is repeated 100 times and the average values are calculated.

The tests evaluate the influence of the most important parameters in the proposed algorithm (Table III). Initially, a random population of 5 rules is generated to seed the algorithm. The genetic process is repeated for 50 generations; that is, 50 cycles of reproduction, evaluation, selection and death.

For the evaluation, the average on the 100 runs of the following measurements are assessed:

- **Diagnosis Error Rate**: proportion of incorrect diagnosis over the total number of problems. This measurement counts neither the problems that are not diagnosed (i.e.: are detected as being normal) nor the false positives. It is given by:

$$E_d = \frac{N_{de}}{N_p} \tag{4}$$

  Where N_{de} is the number of problems diagnosed as a different problem and N_p the total number of problems (1500 in the case of these tests). This measurement indicates the accuracy of the classifier.
- **Undetected Rate**: proportion of problems that are not diagnosed at all. It is given by:

$$E_u = \frac{N_{un}}{N_p} \tag{5}$$

  Where N_{un} is the number of problematic cases that are classified as normal. This measurement indicates the reliability of the fuzzy controller. It indicates its ability to actually detect a problem.
- **False Positive Rate**: proportion of normal cases that are diagnosed as a problem. Given by:

$$E_{fp} = \frac{N_{fp}}{N_n} \tag{6}$$

  Where N_{fp} is the number of normal cases diagnosed as having a problem and N_n is the total number of normal cases (3500 in the case of these tests). A high False Positive Rate indicates that the fuzzy logic controller is not very usable; since there is a high chance of false alarms. Note that even a relatively low E_{fp} can be translated into a high absolute number of false positives in the output, since the number of normal cases is usually much higher than the number of problems.

TABLE III
PARAMETER VALUES

Test	Variable	Default value	Tested values
1	α	24	6, 12, 18, 24, 30, 36, 42, 48, 54, 60
2	MID	0.2	0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9
3	MDA	0.4	0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1
4	reproduction ratio	0.5	0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1
5	MMF	5	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
6	Training cases	Only problematic	Only problematic, problematic + normal
7	Expert elicited rules		Learned rules, expert rules
8	Live network example		Default and Enhanced (α=27,MID=0, number of initial rules = 20, MMF=20)
9	Scalability test		Default and Enhanced (α=42,MDA=0.1, loops = 100)

Note that $E_p = E_d + E_u$ constitutes the total error rate over the input problems (that is, the probability that a problematic case in the input of the FLC produces a wrong diagnosis in the output). The overall error (that is, the probability that a specific diagnosis is wrong) is given by:

$$E = P_n \cdot E_{fp} + P_p \cdot E_p \tag{7}$$

Where P_n and P_p are respectively the proportion of normal and problematic cases in the validation set. Nevertheless, the results given in Section IV-C are given as the three separate terms described earlier, because each one reflects an important characteristic. Throughout the study of the results, it will be considered that there is a previous detection stage that separates normal cases from problematic cases, thus, the main objective will be to minimize E_p. There is a trade-off between the Undetected Rate and the False Positive Rate. If there is no previous filter to detect problems (i.e. the detection also relies on the fuzzy controller, along with the functionality of diagnosis), it is very important to minimize the Undetected Rate. On the other hand, this will normally increase the False Positive Rate. Considering that in a normal scenario, the number of cases showing a normal state outnumber those showing problems, the absolute number of false positives may be higher than the number of correct detections. The probability that a given positive diagnosis is a false positive (the complementary of the Positive Predictive Value) is given by:

$$P_{fp} = 1 - PPV = \frac{P_n \cdot E_{fp}}{P_n \cdot E_{fp} + P_p \cdot (1 - E_u)} \tag{8}$$

C. Results

Test 1: α

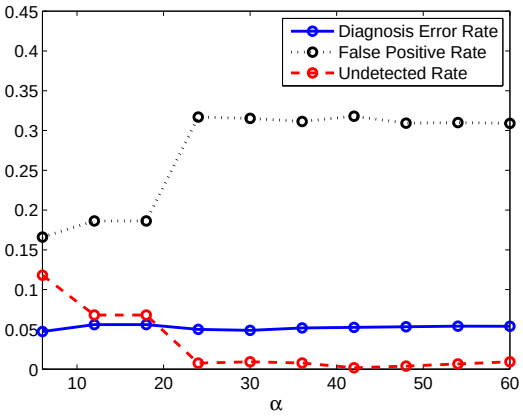

Fig. 9. Error rates for variable α.

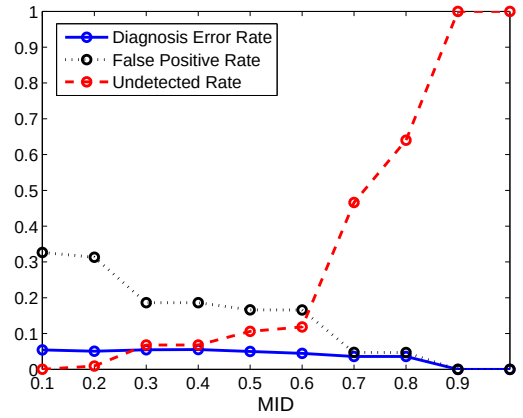

Fig. 10. Error rates for variable MID.

This experiment finds the influence of α. This variable regulates the sensitivity of the algorithm to rare cases. A small value of α gives a low score to rules that cover uncommon cases. A higher value lets the score of a rule grow rapidly as its base increases.

The Diagnosis Error Rate, Undetected Rate and False Positive Rate are depicted in Fig. 9.

The Diagnosis Error Rate is not very sensitive to the value of α. The False Positive Rate increases with α, whereas the Undetected Rate decreases with α. A value of α between 20 and 30 provides a good compromise in the values of both the Undetected Rate and the False Positive Rate. However, if minimizing the Undetected Rate is considered more important than minimizing the false positive rate, then a high value of α should be selected.

Test 2: MID

The MID parameter determines the minimum score that a rule should have to be fused with other rules to produce more general rules. The results are shown in Fig. 10.

The Diagnosis Error Rate and False Positive Rate decrease as MID increases. For $MID = 0.9$, both error rates are zero. Nevertheless, the Undetected Rate is 1, which means that absolutely no problem is diagnosed; there is no actual diagnosis system, as shown in Fig. 11, where the average

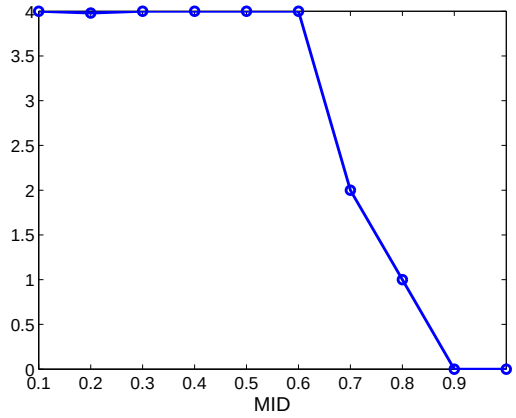

Fig. 11. Number of output rules for variable MID.

number of rules is depicted. As MID increases, the number of rules decreases. A high MID is more restrictive, thus including less rules in the final rule set. For $MID = 0.9$ there is no rule with the required score to be included in the final rule set, so nothing is diagnosed. There is a direct relation between the number of rules and the Undetected Rate. Likewise, as the number of rules decreases, the False Positive Rate also decreases, as less rules produce a smaller chance of a wrong diagnosis. Given the values $P_n = 0.7$, $P_p = 0.3$, and taking the values of E_{fp} and E_u for the default value of $MID = 0.2$ (0.3179 and 0.0075, respectively), according to Eq. (8), $P_{fp} = 0.4277$. That is, 42.77 percent of the times, when the system indicates positive, it is a false alarm. This is for a network where 30 percent of the sectors have problems. For a better network, that number would increase. This stresses the need of a detection phase that filters normal cases, avoiding their analysis by the diagnosis fuzzy logic controller. Supposing that an ideal detection phase is used, $P_n = 0$ and $P_p = 1$ by definition. Applying Eq. (8), $P_{fp} = 0$.

Anyway, a detection phase might still have a small error rate that lets some false positives pass, so it still makes sense to try to keep a low False Positive Rate, even though this criterion is not the priority. Since the Diagnosis Error Rate is relatively insensitive to the variation of MID, the main criterion will be to minimize the Undetected Rate. As a conclusion, to keep a low Undetected Rate, MID should be low (around 0.2).

Test 3: MDA

This experiment evaluates the influence of the MDA parameter over the error rate. This parameter regulates the minimum degree of truth of an antecedent for a case to consider it covered. This modifies the base of the rules, and consequently their scores. Fig. 12 depicts the results.

Again, the tradeoff between undetected rate and False Positive Rate is observed. Since a higher MDA is more restrictive, the effect in the number of rules is similar to the case of MID, as Fig. 13 shows.

The best value of MDA depends on the priorities when tuning the algorithm. To obtain a minimum Undetected Rate, the value for MDA should be lower or equal to 0.3. Nevertheless, for values 0.1 and 0.2, both the Diagnosis Rate and False

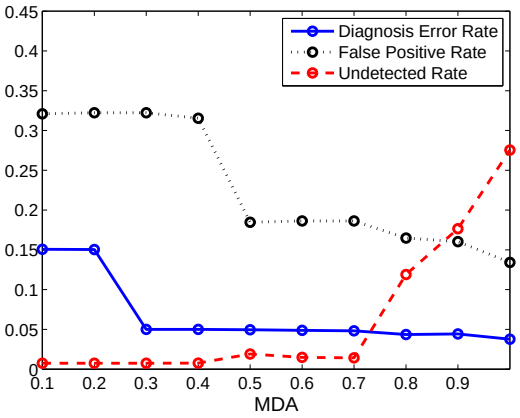

Fig. 12. Error rates for parameter MDA.

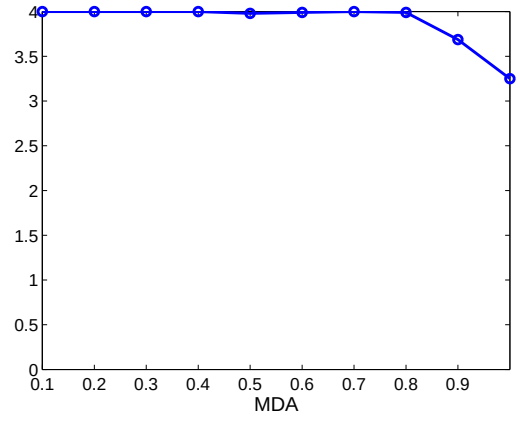

Fig. 13. Average number of rules for MDA parameter.

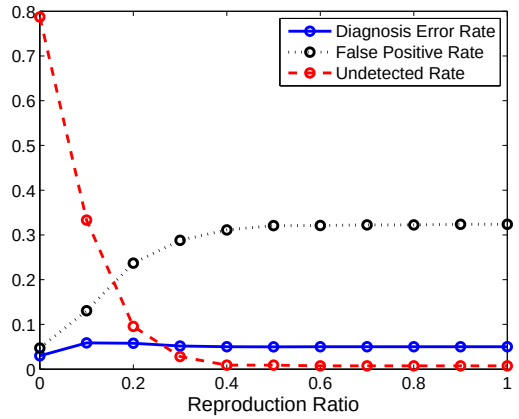

Fig. 14. Error rates for the reproduction ratio.

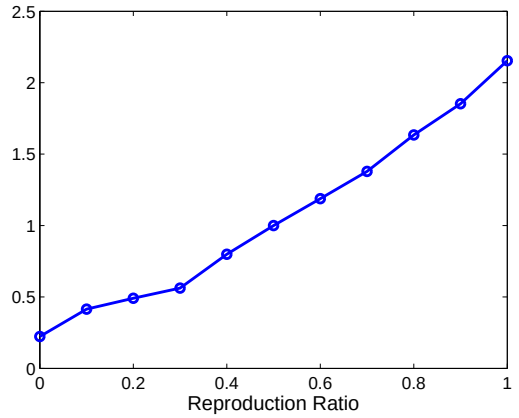

Fig. 15. Relative execution times for different values of the reproduction ratio.

Positive Rate are at their maximum. For 0.3, the Diagnosis Error Rate decreases to almost a third of its maximum value. Therefore, 0.3 is a good choice that has a low Undetected Rate and a relatively low Diagnosis Error Rate. To obtain a low False Positive Rate a good choice would be a value of 0.5 for MDA, which has a slightly worse Undetected Rate. Since the addition of a detection stage would eliminate (or at least reduce) the number of normal cases in the input of the FLC, the False Positive Rate would no longer be an important factor. The Undetected Rate would be transformed into the proportion of cases that are known to be problematic, but cannot be diagnosed. Therefore, the minimization of the Undetected Rate should be prioritized over the minimization of the False Positive Rate.

Test 4: Reproduction ratio

The reproduction ratio adjusts the probability of reproduction for each rule. A high value of this parameter increases the number of combinations tested in the algorithm, at the cost of an increased execution time. In Fig. 14 the measured errors are represented. The execution time with different reproduction ratios relative to the execution time taken with the default configuration is shown in Fig. 15.

The results show that for a low reproduction ratio (lower than 0.5), the Undetected Rate decreases as the reproduction ratio increases. At the same time, the False Positive Rate increases. For values higher than 0.5, all the rates are stagnant. On the other hand, the relative execution time increases as the reproduction ratio grows. Therefore, a value of 0.5 for the reproduction rate is the optimal, since it offers results as good as those obtained for higher ratios, without increasing the execution time.

Test 5: MMF

The MMF adjusts the number of rules that are created each time that a rule reproduces by mutation. A high value of MMF provides an increased exploration capacity, but also increases the execution time as it adds rules to the system. Fig. 16 depicts the error rates as a function of the MMF and Fig. 17 shows the relative execution time as compared to the configuration with a default value for MMF ($MMF = 5$).

As the number of mutated offspring grows, the algorithm produces better results. The Diagnosis Error Rate is more or less stagnant all along the executions. The Undetected Rate shows a large decrease as the number of mutated offspring grows, and the False Positive Rate grows at the same time. The increased exploration provided by this method greatly influences the results.

For values of MMF higher than 6, the Undetected Rate

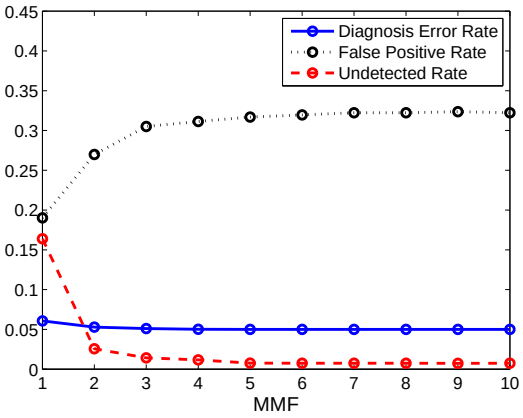

Fig. 16. Error rates for MMF.

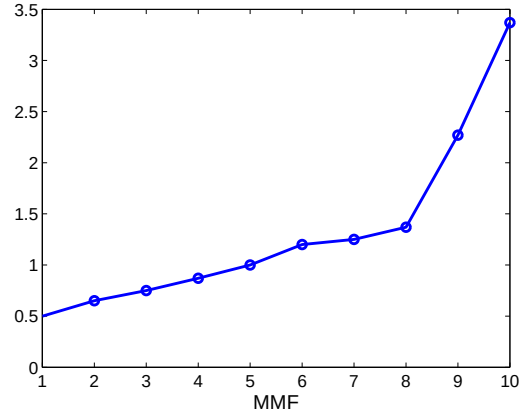

Fig. 17. Relative execution times for different configurations of MMF.

reaches its minimum. Therefore, for values lower than 6, the higher MMF is the better the solution, at the cost of computing time. For values higher than 6, the quality of the solution does not improve, despite the increase of execution time.

Test 6: Training using normal cases

In all the previous tests, the training was done using only cases that had problems. In this experiment, the training will be done with the default values of all parameters (Table III), but using normal cases in the training phase together with the problematic cases. Also, to isolate the effects of α, the default value is used when not using normal cases, and $\alpha \cdot \frac{N_p+N_n}{N_p}$ when using normal and problematic cases in the training. When using this modified value of α in Eq. (3), the absolute number of cases c that a rule must cover in order to obtain a certain base B is the same. Therefore, the variations in the results are only dependent on the fact of using normal cases in the training. The results are depicted in Fig. 18.

The Diagnosis Error Rate and Undetected Rate suffer a slight degradation, whereas the False Positive Rate is clearly improved. This is due to the the fact that the algorithm evolves the rules with knowledge about normal cases. Therefore, it will be less likely that rules that classify normal cases as problematic are considered valid. Thus, the rules are more

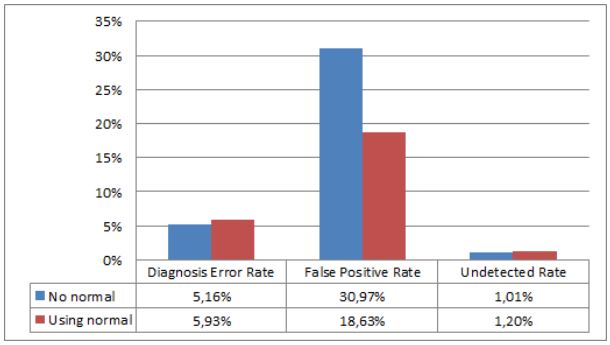

Fig. 18. Results comparing the presence and lack of normal cases in the training set.

evolved towards avoiding false positives. The improvement of the False Positive Rate comes at the cost of an increased execution time, due to the increase in the number of individual applications of rules over cases when the evaluation in the genetic loop is done (appr. 3.5 times higher execution time when including normal cases). When no normal cases are used in training, each individual rule is evaluated on 30 percent of 2000 cases (600), and when normal cases are used, each rule is evaluated over all the cases. The decision of using normal cases in training or not will depend mostly on whether there will or will not be a preliminary detection phase. In case there is, since false positives are not a problem, the training should be done without normal cases, because of the reduced execution time and slightly better error rates. On the other hand, if no detection phase is used, reducing the False Positive Rate will result in a more usable diagnosis algorithm.

Test 7: Comparison between expert elicited and learned rules

Some experts were requested to manually define the rules relating the selected causes and symptoms (Table IV). In this experiment, the three error measurements were compared for the expert elicited rules versus the learned rules using the genetic algorithm (default parameters). Results are shown in Fig. 19.

TABLE IV
EXPERT ELICITED RULES

Acc.	Ret.	HOSR	RSRP	RSRQ	Cause
LOW	LOW	HIGH	HIGH	HIGH	SW Problem
	LOW		LOW	HIGH	Coverage
LOW	LOW	HIGH	HIGH	LOW	Quality
	LOW	LOW			Mobility

It can be seen that there are significant reductions in the Undetected Rate when using learned rules. On the other hand, False Positives are much greater when using learned rules, while Diagnosis Error Rate is similar. It can be concluded that the parameters used in these tests favor the reduction of the Undetected Rate. This might be an advantage depending on the requirements of the final system. In any case, the learning algorithm serves its purpose of obtaining a rule set without the direct intervention of experts, which is an advantage, as exposed in Section I.

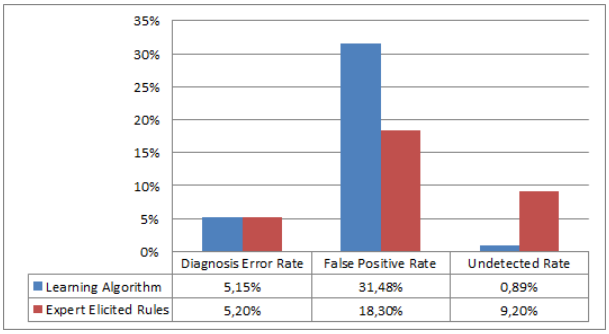

Fig. 19. Results comparing expert elicited and learned rules.

TABLE V
RESULTS OF THE LIVE NETWORK TESTS

Test	Diagnosis Error Rate	Undetected Rate	False Positive Rate
Default parameters	0 %	62.41 %	10.08 %
Enhanced parameters	0 %	19.19 %	21.67 %

Test 8: Live network example

A small test with 72 live network cases was carried out following the same steps as with the simulated cases. Although there were not enough real examples for an in-depth analysis such as in the previous tests, a small amount of available cases may show the behavior of the presented learning algorithm in a real network. The cases are organized in 4 categories: *missing neighbor*, *interferenece*, *high CPU usage* and *normal behavior*. There are 18 vectors representing problems in each category. The PIs used for diagnosing these problems are *accessibility*, *retainability*, *HOSR*, *average RSSI* and *number of CPU overload alarms*. A cross validation technique is used where each category (except *normal*) is divided in two random partitions. One of these partitions is aggregated with partitions of other categories to create a training set, and the remaining partitions, along with the normal cases are used as a validation set. The algorithm is trained with the training set and then the error measurements are taken with the testing set, following the methodology used with the simulated cases. This process is repeated 100 times and the averages of the errors are extracted. The results of the test are shown in Table V.

Using the default values for the parameters, the obtained results are very poor. This is mainly due to the small number of available cases. This has a complex effect on the evolution of the population. Since the number of rules is low, a rule with a low success rate has less probabilities of surviving. Therefore, bad rules die earlier. This reduces the initial population rapidly, and, therefore, the gene pool, that is, the reserve of *"not-so-good"* rules that may produce very successful rules via a small mutation. In this population bottleneck, only the best rules survive, and they may not cover all the cases. Since the best rules have a lower probability of mutating, there is a lower chance that new rules covering the gaps in the training sets appear, and therefore, the Undetected Rate increases. To fix this, it is important that the algorithm has the opportunity of exploring many possibilities. Therefore the immediate actions that can be taken are increasing the MMF to increase the exploration. Also, in this situation of a small gene pool, it will help to increase the chances of exploring rules that would otherwise be ignored. Therefore, the number of initial rules is also increased. A new set of enhanced parameters is created where the number of initial rules is 20 and the MMF is 10. The results are clearly improved, although still the Undetected Rate is higher than in the tests due to the very small number of training cases.

TABLE VI
ADDITIONAL PI MODELS. PDF OF EACH PI CONDITIONED TO AN EXISTING PROBLEM

PI	Type		Parameters/Cause				
			Nor	SW	Cov	Qual	Mob
SINR	norm	avg	15	13.6	15	7	14.2
		σ	0.8	1.4	1	1.7	1
Dist.	norm	avg	0.9	0.9	1.1	1	0.9
		σ	0.2	0.3	0.3	0.2	0.3
Thp.	norm	avg	11.4	7.5	7.9	6	9.7
		σ	3.1	4.25	6.1	5.25	4.25
CPU	negative binomial	n	0.052	2.07	0.052	0.024	5.5e-3
		p	0.01	6.3e-3	9.6e-4	4.7e-4	2.3e-3
RSSI	norm	avg	-116.7	-112	-114	-109	-119
		σ	0.5	10	10	11	12

Test 9: Scalability test

In all these tests, we have used 5 PIs to diagnose the problems. In real networks, the number of PIs is usually much larger, and the interactions much more complex. Therefore, it would be useful to know if the proposed algorithm can scale up and manage an increased number of PIs. To test this, an extended dataset has been created, containing the same problems as the default dataset, and adding 5 PIs to the original set of modeled PIs:

- 95 percentile of SINR: value of SINR under which 95 percent of the samples fall. It is considered *low* when the value is below 10 dB and *high* when it is above 12 dB.
- 95 percentile of the distance: radius of the circumference that encompasses 95% of the users of the sector. Distances below 1 Km are *low*, whereas distances above 1.1 Km are *high*.
- Throughput: Data rate that the sector successfully transmits to the users (downlink). Throughput values below 7 Gbps are *low* and values above 7.5 Gbps are *high*.
- Number of CPU overload alarms: Number of alarms per hour indicating that the CPU of the sector is overloaded. Below 20, the values are considered *low* and above 40, *high*.
- Average RSSI: Total signal power received in the operational bandwidth. Below -110 dBm, it is considered *low*, and above -108 dBm, *high*.

The model parameters of these PIs are summarized in Table VI. The model parameters and the requirements are all based on expert input.

The configuration of the training and testing sets are kept on the same values as described in Section IV-B, and a test with the default parameters is executed. The results are shown in Table VII.

TABLE VII
RESULTS OF THE SCALABILITY TESTS

Test	Diagnosis Error Rate	Undetected Rate	False Positive Rate
Default parameters	0.6 %	50.13 %	3.26 %
Enhanced parameters	1.2 %	1.4 %	4.2 %

A degradation in the performance of the obtained rule set is clearly observed. This is due to more complex individuals (rules) involved in the evolutionary process. Since the Undetected Rate has grown, it can be concluded that not enough rules are being created to cover all cases. Therefore, a new set of improved parameters is tested. The number of loops is increased from 50 to 100, to ensure that the evolutionary process has enough time to process the increased complexity of the individuals. The α parameter is increased from 24 to 42, so that the sensitivity to unusual cases is increased. Finally, the MDA parameter is reduced to 0.1, so that lower scored rules survive to have a chance of improving. With these parameters, the performance is largely improved (Table VII). In fact, these results improve the Diagnosis Error Rate over the default number of PIs, since there is more information for the FLC to process and discriminate among the diagnoses. Also, there is a reduction in the False Positive Rate due to the increase in the number of variables, that reduces the probability that a normal case has the KPI values that are typical of a specific problem. Nevertheless, this gain comes at a cost of execution time and a slight increase in the Undetected Rate. Although the implementation of the algorithm is not speed-optimized, the new data set takes more than 36 times more time than the default data set.

V. CONCLUSIONS

A novel approach for obtaining the rule base of fuzzy controllers for mobile network self-diagnosis has been presented, using a learning method instead of direct knowledge acquisition. The proposed method is a supervised, genetic learning algorithm, whose inputs are vectors of PIs, alarms and configuration parameters labeled with the fault cause.

The method has been evaluated using a reduced model comprising some fault causes and PIs.

The experiments cover the main configuration parameters of the algorithm and the presence or absence of normal cases in the training set. The conclusions are that the parameters must allow diversity in the rule base; that is, they must not filter rare cases or cases that loosely follow a rule. If the parameters are too restrictive, the output rule base is unreliable, in the sense that it will often be unable to diagnose (or even detect, if the fuzzy controller is assigned this task) a proportion of the problems. On the other hand, the increase in reliability comes at a cost; the fuzzy rule set will detect some normal cases as problems. This problem is solved if there is a detection stage that filters out normal cases prior to its analysis by the diagnosis stage. Results show that the proposed method is quite robust to the selection of the parameters, since there are only slight variations in the diagnosis error for different values of the parameters. The selection criterion should be more dependent of the Undetected Rate, since the impact of the parameters on this measurement might be high. Nevertheless, the method is still robust, since the allowed parameter ranges for low Undetected Rate are broad.

ACKNOWLEDGMENT

This work has been partially funded by Optimi-Ericsson, Junta de Andalucía (Agencia IDEA, Consejería de Ciencia, Innovación y Empresa, ref. 59288; and Proyecto de Investigación de Excelencia P12-TIC-2905) and ERDF.

REFERENCES

[1] NGMN, *Use Cases Related to Self-Organising Network, Overall Description*, Next Generation Mobile Networks (NGMN) Alliance, http://www.ngmn.org, April 2007.

[2] 3GPP, *Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Universal Terrestrial Radio Access Network (E-UTRAN); Overall description; Stage 2*, ts 36.300 ed., Next Generation Mobile Networks (NGMN) Alliance, December 2012.

[3] ——, *Telecommunication management; Self-Organizing Networks (SON); Concepts and requirements*, ts 32.500 ed., Next Generation Mobile Networks (NGMN) Alliance, September 2012.

[4] R. Barco, P. Lázaro, and P. Muñoz., "A unified framework for self-healing in wireless networks," *IEEE Communications Magazine*, vol. 50, pp. 134–142, December 2012.

[5] 3GPP, *Telecommunication management; Self-Organizing Networks (SON); Self-healing concepts and requirements (Rel 11)*, ts 32.541 ed., Next Generation Mobile Networks (NGMN) Alliance, September 2012.

[6] M. Asghar, S. Hamalainen, and T. Ristaniemi, "Self-healing framework for LTE networks," in *IEEE 17th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, September 2012.

[7] G. A. Barreto, J. C. M. Mota, L. G. M. Souza, R. A. Frota, and L. Aguayo., "Condition monitoring of 3G cellular networks through competitive neural models," *IEEE Trans. Neural Networks*, vol. Vol.16(5), pp. 1064–1075, 2005.

[8] R. Barco, P. Lázaro, L. Díez, and V. Wille., "Continuous versus discrete model in autodiagnosis systems for wireless networks," *IEEE Transactions on Mobile Computing*, vol. Vol.7 (6), pp. 673–681, June 2008.

[9] R. Barco, P. Lázaro, V. Wille, L. Díez, and S. Patel, "Knowledge acquisition for diagnosis model in wireless networks," *Expert systems with Applications.*, vol. Vol.36, Issue 3, Part 1, pp. 4745–4752, April 2009.

[10] Univerself, "Univerself project," http://www.univerself-project.eu/, 2012.

[11] L. Bennacer, L. Ciavaglia, A. Chibani, Y. Amirat, and A. Mellouk, "Optimization of fault diagnosis based on the combination of bayesian networks and case-based reasoning," in *IEEE Network Operations and Management Symposium (NOMS)*, April 2012.

[12] C. Hounkonnou and E. Fabre., "Empowering self-diagnosis with self-modeling," in *8th international conference Network and service management (CNSM), and 2012 workshop on systems virtualization management (SVM)*, October 2012.

[13] COMMUNE, "Commune (COgnitive network ManageMent under UNcErtainty)," http://projects.celtic-initiative.org/commune/, 2012.

[14] C. Project, "Specification of knowledge-based reasoning algorithms," 2012, deliverable 4.1.

[15] P. Szilagyi and S. Novaczki, "An automatic detection and diagnosis framework for mobile communication systems," *IEEE Transactions on Network and Service Management*, vol. 9, no.2, pp. 184–197, June 2012.

[16] M. Amirijoo, L. Jorguseski, T. Kürner, R. Litjens, M. Neuland, L. C. Schmelz, and U. Türke, "Cell outage management in LTE networks." in *6th International Symposium on Wireless Communication Systems (ISWCS)*, 2009.

[17] H. Eckhardt, S. Klein, , and M. Gruber., "Vertical antenna tilt optimization for lte base stations." in *IEEE 73rd Vehicular Technology Conference (VTC Spring)*, 2011.

[18] R. Razavi, "Self-optimisation of antenna beam tilting in LTE networks.." in *IEEE 75th Vehicular Technology Conference (VTC Spring)*, 2012.

[19] L. A. Zadeh, "Fuzzy sets," *Information and control*, 1965.

[20] P. Muñoz, R. Barco, and I. de la Bandera, "On the potential of handover parameter optimization for self-organizing networks," *IEEE Transactions on Vehicular Technology*, vol. 62, no.5, pp. 1895–1905, June 2013.
[21] ——, "Optimization of load balancing using fuzzy Q-learning for next generation wireless networks," *Expert Systems With Applications*, vol. 40, no. 4, pp. 984–994, March 2013.
[22] C. Lee, "Fuzzy logic in control systems: fuzzy logic controller," *IEEE Transactions on I. Systems, Man and Cybernetics*, pp. 404–418, 1990.
[23] S. Kotsiantis, "Supervised machine learning: A review of classification techniques," *Informatica*, vol. 31, pp. 249–268, 2007.
[24] N. Grira, M. Crucianu, and N. Boujemaa, "Unsupervised and semi-supervised clustering: a brief survey," *A review of machine learning techniques for processing multimedia content, Report of the MUSCLE European Network of Excellence (FP6)*, 2004.
[25] O. Cordon, F. Herrera, and P. Villar, "Generating the knowledge base of a fuzzy rule-based system by the genetic learning of the data base," *IEEE Transactions on Fuzzy Systems*, vol. 9, no. 4, pp. 667–674, August 2001.
[26] A. Gonzalez and R. Pérez, "SLAVE: A genetic learning system based on an iterative approach," *Fuzzy Systems, IEEE Transactions on*, vol. 7, no. 2, pp. 176–191, April 1999.
[27] S. Guillaume, "Designing fuzzy inference systems from data: an interpretability-oriented review," *Fuzzy Systems, IEEE Transactions on*, vol. 9, no. 3, pp. 426–443, June 2001.