Paper

An Eclectic Approach to Network Service Failure Detection Based on Multicriteria Analysis with an Example of Mixing Probabilistic Context Free Grammar Models

Paweł Białoń

Abstract—A method of failure detection in telecommunication networks is presented. This is a meta-method that correlates alarms raised by failure-detection modules based on various philosophies. The correlation takes into account two main characteristics of each module and the whole meta-method: the percentage of false alarms and the percentage of omitted failures. The trade-off between them is tackled with aspiration-based multicriteria analysis. The alarms are correlated using linear classification by support vector machines. An example of the profitability of correlating alarms in such way is shown. This is an example of probabilistic context free grammars (PCFGs), used to model the proper runtime paths of network services (and thus usable for detecting an improper behavior of the services). It is shown that the linearly mixing PCFGs can add context handling to the PCFG model, thus augmenting the capabilities of the model.

Keywords— failure detection, linear separation, probabilistic context free grammars, support vector machine.

1. Introduction

The domain of automatic failure detection in telecommunication and computer networks, becoming an extensively exploited domain, distinguishes with an exceptional variety of approaches used [1]. A broad spectrum of statistical/stochastic methods are used in failure detection, so are signal processing, discrete time sequence analysis, finite state machine methods, automatic reasoning, data mining, various classifiers, e.g., based on neural nets. The multitude of the existing approaches best proves that none of them is perfect. By choosing one of them, a designer of a network monitoring tool has to strongly narrow the area of a successful application of the tool.

This paper presents a concept of a meta-tool capable of integrating very different approaches known from the literature.

The proposition assumes an open architecture of the proposed tool – new literature approaches could be implemented as new modules of the tool. The indications of various modules are correlated, yielding a much more reliable assessment of the network state. Interesting is the way of correlating indications obtained from modules of completely different philosophies. The correlation uses linear classification and multicriteria analysis (we describe each module with two criteria that seem to be common throughout various approaches: the percentage of overlooked failures and the percent of false alarms). Several auxiliary hard optimization and simulation problems: large-scale, nondifferentiable, nonconvex are obtained. We propose to simplify some of them before solving, using statistical methods.

A fundamental question arises whether it is reasonable and useful to make linear combinations of outputs from various detection procedures. These procedures themselves may be described in languages strongly differing from "linear combining" – like some discrete approaches. To support our approach, we use a very interesting example. We mix indication from two modules detecting failures based on probabilistic context free grammar (PCFG) analysis of runtime paths [2]. By mixing them we essentially enlarge the expressiveness which a single module had: we add a context to the used grammar!

We have to make the area of application of our proposition more precise. It includes an automatic *failure detection*, where the management tool signalizies that a failure of the network is present and possibly gives some rough information of a type of the failure. The presented methodology can be applied to detecting both *service failures* and strict *network failures*. Though the paper more precisely analyzes some service failure approaches, we will refer jointly to both the types of failures using the short term of "network failures". Also, our tool would be suitable for a broader domain of *anomaly detection*, where an anomaly is understood in a broader sense that a failure (hardware, network-software or network-service) can express also untypical user behavior, connected with malicious activities, possible intrusions, frauds. Switching to making *proactive* failure of anomaly detections would be possible, by making some simple technical extensions, like shifting relevant time sequences within the tool, during its learning phase. However, there are bold challenges of *failure localization* (*reasoning about the failure reasons*) and *automatic* or *semi-automatic failure repairing* in which our tool would not acquit itself well. These are tasks by nature not

well-suited for supervised learning methods, to whom our method ranks. These tasks are solved by expert systems and other artificial intelligence approaches.
In Section 2 we shall try to show the variety of existing failure detection approaches. The structure of our tool and the complex problem of tuning it will be discussed in Section 3. A discussion of the soundness of the described approach, together with the conclusions from our work is given in Section 4.

2. The Variety of Detection Approaches

We shall give a flavor of the variety of existing literature approaches to automatic network failure detection.
Various methods use various data about the network, coming from various sources. Let us, however, stress the increasing role of the simple network management protocol – SNMP (see [1] and the references therein) in acquiring such data. The network state is described by several tens of thousands variables (so called management information base variables – MIB variables), which can be probed at regular time bases. SNMP delivers variables connected with the traffic in particular network arcs, TCP/IP (transmission control protocol/Internet protocol) information (traffic, number of open connections, number of packets accompanying opening and closing connections, or accompanying some errors). Other useful sources of data are system logs, e.g., regarding line commands given by users (the logs are used in intrusion detection).
Let us enumerate some important classes of methods used in automatic failure detection.

1. Many approaches base on signal processing methods and stochastic methods. Usually, data about the network traffic in various moments are time sequences, often treated as realizations of stochastic processes.

   Such approaches can be often depicted as systems with elements like a filter, a statistic estimator, a discriminator, an alarm generator. The “systems” more or less accurately follow the structure from Fig. 1.

   An *alarm* is understood as a warning about incorrectness of the work of the network. An alarm is, however, easy to obtain and a rose alarm cannot itself prejudge that a failure is present. Alarm generation can be merely caused by exceeding some (lower or upper) limit value of the traffic intensity in some network arc. Similarly, the excess of some error frames rate may be examined and, in more advanced methods, the excess of some thresholds of certain signal statistics, calculated by an estimator.

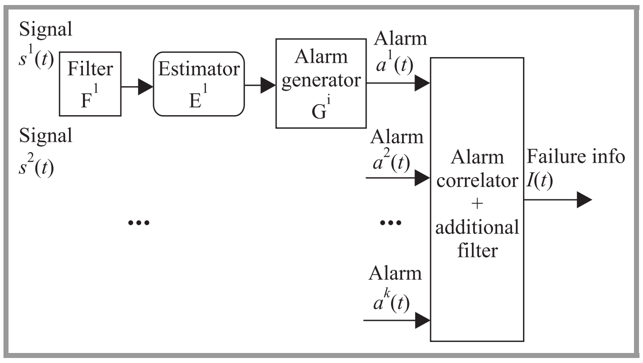

Fig. 1. A typical structure of the tools based on alarms generated from continuous traffic time sequences.

   A single alarm, when obtained in a simple way, is not very reliable in detecting failures (e.g., it may be false). Usually failures cause several alarms (e.g., simultaneously, a decrease of the traffic intensity and an increase of the error rate). The *alarm correlator* is an element obtaining the information on the presence of particular alarms and, based on it, deciding whether a failure is present (or localizing the failure – in systems that are capable of doing it). In particular, an occurrence of a single alarm at a time, can be ignored by the correlator.

   A pure value of some signal at some time may be not very relevant in raising or correlating alarms. For example, we would probably want to ignore some short-term incorrectnesses of the signal. Thus the described systems are often equipped with numerous *filters*, transforming signals both before and after the alarm generation.

   An example of the class of methods being discussed is presented in [3]. The traffic intensity in a certain network arc plays the role of signal $s^i(t)$ at Fig. 1. This signal is filtered (integrated within some time window) to reject temporary incorrectnesses. The obtained integral is some stochastic process; its distribution is modeled and estimated on a simple basis of ranking its values into several predefined intervals. When the signal goes out of a certain confidence interval (the intervals are different for various times of a day), an alarm is generated (if the situation is not only temporary – one more low-pass filter is applied).

   An interesting approach is presented in [4]. Each input signal s^i corresponds to a different MIB variable, usually representing traffic in a different layer: TCP, IP, data link. The idea is that anomalies propagate through the network layers, thus should be observable from different variables. Filter F is an autoregressive (AR) filter of rank 1. The estimator, in turn, calculates the defection of some simple statistics of the filter output (which are based on the variance) from the reference statistics (obtained from observations made in the immediate past – in some time window spreading up the present). A big defection means an “abrupt change” in the statistic properties of the input signal and thus – a probable occurrence of a failure. The volume of the defection corresponding to input s^i is expressed by a continuous alarm a^i from interval $[0,1]$. (Value 1 means the strongest defection). So we have continuous alarms, instead of zero-one. Correlating alarms is done by calculating

$$aBa^T ,$$

where $a = \begin{bmatrix} a^1 \\ a^2 \\ \dots \\ a^k \end{bmatrix}$, B is a symmetric $k \times k$ matrix. When this value exceeds some threshold, a failure is ascertained. The presence of matrix A, with $\mathcal{O}(k^2)$ coefficients, already allows to model quite complex correlations between the alarms.

2. An example of failure detection by discrete methods is given in [5]. The authors have a method of discovering dependencies between two or more discrete time sequences, called multi-stream dependence detection (MSDD). This method can be applied to the problem of network failure proactive detection. The evolution of the network state (some input signal or some set of trivially obtained alarms) is described as a sequence of discrete values, corresponding to consecutive time moments, for example,

   ```
   1 5 7 3 6
   ```

   We can have two training sequences: one representing the presence or absence of a failure, another – some network signal. Then foreseeing a failure corresponds to finding correlation between elements of the sequences.

   Finding correlation in MSDD bases on templates, e.g., `*` in a template means "any value". Continuing our example, in the first sequence let us denote a presence of a failure by `F`, a normal network state – by `N`. Let us have positive integers in the second sequence, coding the values of some quantity measured in the network. MSDD can, for example, find a rule of the form:

   ```
   N N N N N N N N N F
   * 2 1 * * * * * * *
   ```

   The rule says that, the occurrence of the event consisting in an immediate (in one time instance) transition of the observed quantity from 2 to 1 indicates a presence of failure in 7 time units.

   Certainly, also more realistic, more complicated rules can be obtained. The algorithm searches for rules that have an outstanding support in the training data. The algorithm uses a mere Bayesian apparatus.

3. While the researchers prefering methods of signal analysis concentrate more on the alarm generation, the artificial-intelligence experts more willingly deal with the alarm correlation and also try to point out the reasons of the failure.

   In many works, like [6], *dependency graphs* are used to describe the propagation of a failure in the network. Some particular elements of the network are distinguished (a particular device, protocol, service, server, etc.). They are drawn as the graph nodes. If a malfunction of element A causes a malfunction of element B with probability p, we draw an arrow on the graph, from A to B, and p is denoted by the arrow. The propagation of a malfunction may be multi-stage, ending with an observed failure. Again, using the apparatus of conditional probabilities we can identify the most probable initial reason of the failure. In the cited paper, the way to do this leads through solving a combinatorial-optimization problem.

   Instead of using graphs, we can use logical expressions of some canonical form, of a lower nesting level [7]. Both the approaches need a laborious phase of obtaining the necessary knowledge from an expert. Both of them need a relatively hard updating of the monitoring software as the network changes in time (e.g., as it grows).

4. Detecting failures in remote databases (and other remote service environments) based on *run-time paths* is described in [2].

   A *runtime path* is composed of events happening in various places of the system. Events must have a common *request identifier* to be included into the same path. For example, an event can be a remote invocation of a procedure, data flow between remote components of the system, realization of a database query, spawning a thread by a Java application, etc. A request identifier can be a session identifier in the hypertext transfer protocol (HTTP).

   The simplest way to validate the correctness of the path is the analysis of the delays between events. The delays can have some reference distributions, build during an observation of a normal work of the distributed system. The conformance of delays currently being measured to these distribution can yield an assessment of the correctness of the system state.

   A much more powerful tool, so called probabilistic context free grammar can assess the correctness of the order of events on a path. PCFG (see [2] and the references therein) is an extension of mere context-free grammars, consisting in defining probabilities of the productions.

   A PCFG is a 5-tuple consisting of:

   – set of terminal symbols: $\mathfrak{T} = \{T^k : k = 1, \dots V\}$;
   – set of nonterminal symbols: $\mathfrak{N} = \{N^i : i = 1, \dots n\}$;
   – starting symbol $S \in \mathfrak{N}$;
   – set $\mathfrak{R} = \{R^j\}$ $(j = 1, \dots p)$ of productions of the form $N \rightarrow s$, where $N \in \mathfrak{N}$ and s is a finite sequence consisting of elements of $\mathfrak{T} \cup \mathfrak{N}$;
   – set of probabilities of productions $\mathfrak{P} = \{P^j\}$ $(j = 1, \dots p)$.

The probabilities of all productions with a given symbol on the left hand side must sum up to 1.

We can try to *derive* a given word (a sequence of terminal symbols) from the grammar, starting with the starting symbol and iteratively applying suitable productions until we end up with the word. For simplicity we assume that the derivation of any derivable word is unique.

Example 1: Let grammar G_1 be defined by $\mathfrak{N} = \{S, X, Y\}$, $\mathfrak{T} = \{a, b, c, d\}$, $\mathfrak{R} = \{$
$S \to XY \ (P = 1)$
$X \to a \ (P = 0.2)$
$X \to b \ (P = 0.8)$
$Y \to c \ (P = 0.5)$
$Z \to d \ (P = 0.5)$
$\}$

The derivation of word bc is following:
$S \to XY \to Xc \to bc$.
We used the 1st, the 3rd and the 2nd productions, in order.

The product of the probabilities of the used productions, 0.4 in the example, is the *probability of the word*. If symbols corresponded to events in our system (and words – to runtime paths) the probabilities of the word could be used to assess the correctness of a path (and of the distributed system state).

5. Certainly, other approaches are present. They may be based on "standard" methods (neural nets, other classification algorithms, clustering methods, the Markov process, etc.). Untypical approaches may prove usefulness. In [3], failures are suspected when the network devices are steering the traffic in a "strange" way, i.e., leading some arcs to an unnecessary saturation. In Fig. 2 the saturation threshold for each arc is 10 units, variables by arc denote the current traffic intensities. Then the configuration of intensities $x_{12} = 1$, $x_{23} = 1$, $x_{13} = 10$ is erroneous, arc $(1, 3)$ is unnecessarily saturated, while a bypass exists through node 2. The reasoning seems simple on this simplistic example presented but becomes more sophisticated when we consider longer by-passes and distinguishes "commodities" in arcs, i.e., parts of the traffic with a particular sender and receiver.

   Another untypical approach uses the machinery of reference point multicriteria analysis [8].

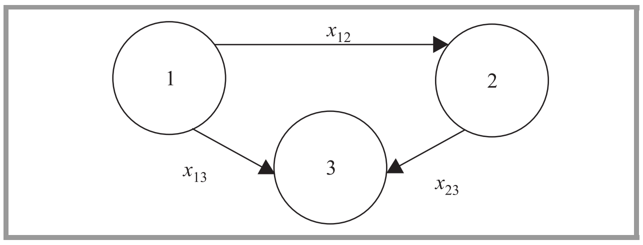

Fig. 2. An incorrectness of the traffic control.

3. The Idea of the Tool

3.1. Structure

Our hypothetical tool will contain failure detection modules of various philosophies. The coexistence of modules is possible due to their uniform treatment in the structure of the tool (Fig. 3).

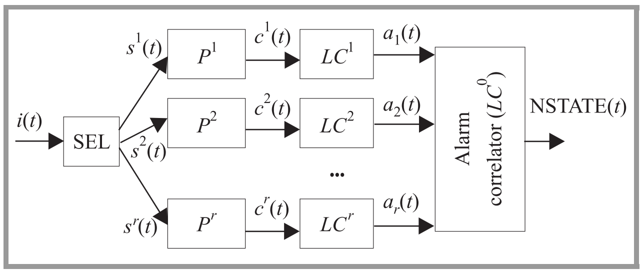

Fig. 3. Structure of the tool.

Each module consists of a *preprocessor*, into which the knowledge of a particular detection approach is coded, and a *linear classifier*. The input vector of the preprocessor (representing the current network state) is transformed to the output vector (preferably, a vector of reals), which describes the network state according to the knowledge of the approach. The output from the preprocessor, in the simplest case, would be the alarm value obtained due to the particular approach but will be rather some multidimensional description of the network state, and the role of the linear classifier would be to transform it into a continuous scalar representing the "normality level" of the network state.

Let us describe the elements of Fig. 3. We shall use a discrete time t – this variable will take natural values. Some of the quantities will be parametrized with this discrete time, thus they may be represented as functions of t. The main elements are:

1. Input signal vector $i(t) \in R^{n_i}$. Its coordinates may come, e.g., from MIB variables collected at time t; in general, they may represent diverse quantities.
2. Selector `SEL` – simple module selecting coordinates of i, from which vectors s^i are formed, used by particular preprocessors.
3. Preprocessors P^i for $i = 1, \dots r$, their respective outputs $c^i \in \mathbb{R}^{\eta_i}$, respective classifiers LC^0, ..., LC^r and alarms $a^i(t) \in \mathbb{R}$ rose by the classifiers for $i = 1, \dots r$. In general, each preprocessor remembers its inputs for at most H time instances, so

$$c^i(t) = c^i(s^i(t); s^i(t-1); \dots s^i(t-H+1)).$$

   There holds $a_i = \phi^i(c(t))$, with (operator ";" denotes the vector/number concatenation), $\phi : \mathbb{R}^{\eta^i}$ being an affine function:

$$\phi^i(x) = \omega^{i\top} x + \gamma^i, \qquad (1)$$

   where $\omega^i \in \mathbb{R}^{H\eta^i}$ and $\gamma^i \in \mathbb{R}$ are the classifier parameters, tuned in the learning phase, described later.

We treat the alarm value of -1 as de facto absence of an alarm, the value of 1 as rising an alarm.

4. Alarm correlator/LC^0. It receives at time t vector $a(t) = (a_1(t),\ a_2(t),\ \ldots,\ a_r(t))$. It returns a number greater than 0 (failure) or not greater than 0 (no failure). We do not equip the correlator with an additional low-pass filter eliminating "short-time failure indications" in our later reasoning, though it is desired in implementations.

3.2. The Case of Many Failures or Future Failures

The presented tool could be only simplistically extended to detect various types of failures, by multiplying the structure from Fig. 3 for separate failures. As already stated, we cannot expect great failure localization possibilities from our tool, which is of supervised learning tools class.

It is easier to augment our tool for the case of proactive failure detection. It suffices for modifying the learning phase, by shifting (in time) the teacher information of the state of the network (failure or no failure).

3.3. Teaching the Classifiers

Finding parameters ω^i γ^i of ith classifier is done in the supervised learning phase. We have historical examples $\mathfrak{e}_j$ $(j = 1,\ldots,n)$ of network states, where $\mathfrak{e}_j$ is an example from a historical time t^j, $e_j = c^i(t^j)$. For every example we know whether it is *positive*, i.e., describes a normal network state, or *negative*, i.e., describes a failure state.

Parameters ω^i γ^i, set to the solution of a certain optimization problem, are called support vector machine (SVM):

$$\underset{\omega^i\in\mathbb{R}^{\eta^i},\gamma^i\in\mathbb{R},y\in\mathbb{R}^n_+}{\text{minimize}} \quad \frac{1}{2}\|\omega\|^2 + Ce^\top y, \tag{2}$$

s. t.

$$-D\cdot(\mathfrak{e}^{j\top}\omega-\gamma)-y+1\le 0 \qquad \text{for } j=1,\ldots n, \mathfrak{e}^j \text{ negative,}$$
$$(\mathfrak{e}^j\omega^\top-\gamma)-y+1\le 0 \qquad \text{for } j=1,\ldots n, \mathfrak{e}^j \text{ positive.}$$

Here $e = (1,1,\ldots,1) \in \mathbb{R}^n$, $C,D > 0$. For the derivation of SVM problems see [9] or [10]. Learning parameter C controls the trade-off between the number of training examples misclassified by the taught classifier and the separation margin.[1] Learning parameter D is augmentation of the problem from [9] to control the relative impact of negative versus positive examples on γ^i and ω^i being found (it will mainly affect the later described trade-off between the tendency of the classifier to raise false alarms and to overlook failures). For a more thorough description of the above problem and a solver capable of solving it, see [11], [12].

[1]The existence of a (big) separation margin causes that the values of ϕ^i for the training examples are (much) isolated from 0. With a big separation margin we can hope for good generalization properties of our classifier.

3.4. Assessing the Modules Performance

Tuning the whole tool will be described in Subsection 3.5. For this, however, we must be able to assess the quality of tunnings (C, D) for a single module. This will be achieved by checking the attained compromise between two criteria: the number of false alarms and overlooked failures by the module. This is important because the whole tool will be assessed by similar criteria.

We divide some historical data about the network state into two sets: training data and test data. We teach a classifier with parameters C, D. We test the so taught module on the test data. We define function $q : \mathbb{R}^2 \to \mathbb{R}^2$ as

$$q(C,D) = \begin{bmatrix} \frac{\text{num. of misclassified positive test examples}}{\text{num. of positive test examples}} \\ \frac{\text{num. of misclassified negative test examples}}{\text{num. of negative test examples}} \end{bmatrix};$$

its coordinates are our *criteria* (the rate of false alarms, the rate of overlooked failures).

We make a parametric experiment – we teach our classifier for various combinations of C, D, i.e., for $(C,D) \in X$, where X is a finite subset of R^2_+. We obtain the following *attainable results set*:

$$Q = \{(y_1,y_2) \in \mathbb{R}^2 :\ \exists_{(c,d)\in X} q(c,d) = (y_1,y_2)\}.$$

We have to reject clearly unnecessary elements of Q, i.e., such elements that the classifier for some other setting of C, D gives one criterion not worse that in this element and the other criterion – better than in this element. By rejecting them we obtain the *efficient results set* (for some particular Pareto order):

$$Q^\star = \{(y_1,y_2) \in Q :\ \neg\exists_{(z_1,z_2)\in Q} \\ (z_1 < y_1 \wedge z_2 \le y_2) \vee (z_1 \le y_1) \wedge (z_2 < y_2)\}. \tag{3}$$

Since during the later tuning of the whole tool will see the modules only in terms of elements of Q (attained results for the module) we shall need to be able to return from an element of Q to (some) setting (C,D) that yielded it. For this reason, the tool must now memorize the relation between the settings and the attained results.

In the further analysis it will be easier to number elements of $Q^\star$ with one variable. Let us number the elements of $Q^\star$ with index ϑ, $\vartheta = 1,2,\ldots|Q^\star|$, according to the growing value of the first coordinate.

Such numbering is not ambiguous: there cannot exist two elements of $Q^\star$ with identical first coordinates and different second coordinates: definition (3) does not allow this.

Now each point in $Q^\star$ may be the value of a function $\varkappa$ of this index:

$$Q^\star = \{(y_1,y_2) = \varkappa(\vartheta) :\ \vartheta = 1,2,\ldots l\},$$

where $l = |Q^\star|$ and $\varkappa : \{1,2,\ldots l\} \to R^2$.

Remark 1: The coordinates of function $\varkappa$ are monotone: $\varkappa_1$ is an increasing function, $\varkappa_2$ is a decreasing function.

The increasing character of $\varkappa_1$ follows from numbering of the elements of Q by the first coordinate. The decreasing

character of $\varkappa_2$ follows from the numbering and from the constriction (3) of set $Q^\star$.
Finally, each module is characterized with its function $\varkappa$.

3.5. Teaching the Alarm Correlator

The alarm correlator yields $\mathtt{NSTATE}(t)=\omega^\top(a^1(t), a^2(t), \dots a^r(t))+\gamma$, where $\omega \in \mathbb{R}^r$ and $\gamma \in \mathbb{R}$ are tunable parameters, ($\mathtt{NSTATE}(t) \le 0$ means a failure, $\mathtt{NSTATE}(y) > 0$ – no failure).
We have been describing some quantities for a single module/classifier. Since we now have to consider all the modules jointly, we equip these quantities with an additional index i denoting the module number (i will run from 1 to r). So we shall make the following transformation in out notation:

$$l \to l_i,\ \vartheta \to \vartheta_i,\ \varkappa_j \to \varkappa_{i,j},\ C \to C_i,\ D \to D_i.$$

Moreover, C, D and ϑ will be vectors now: $C = (C_1, C_2, \dots C_r)$, $D = (D_1, D_2, \dots D_r)$, $\vartheta = (\vartheta_1, \vartheta_2, \dots \vartheta_r)$.
We introduce function $\mathfrak{q}(t, \omega, \gamma)$ assessing the whole system:

$$\mathfrak{q}(C,D,\omega,\gamma) = \begin{bmatrix} \frac{\text{num. of positive test examples misclassified by correlator}}{\text{num. of positive test examples}} \\ \frac{\text{num. of negative test examples misclassified by correlator}}{\text{num. of negative test examples}} \end{bmatrix}.$$

The author would like to thank Prof. Wierzbicki and Dr. Granat for pointing out importance of assessing the system by such two criteria (defined by the coordinates of $\mathfrak{q}$). The choice of the compromise can be left to the network administrator. In the sequel we shall allow this choice with the apparatus of the reference point methodology [13].

We shall try to minimize both the criteria. To merit the "levels of achievement" in the minimizations we shall introduce a *scalarizing function* $s_{\bar{\bar{\mathfrak{q}}},\bar{\mathfrak{q}}} : \mathbb{R}^2 \mapsto \mathbb{R}$, [13] with parameters $\bar{\bar{\mathfrak{q}}} \in \mathbb{R}^2$ (the vector of so called reservation levels) $\bar{\mathfrak{q}} \in \mathbb{R}^2$ (the vector of so called aspiration levels). The reservation level for a criterion (a coordinate of $\mathfrak{q}$) is defined as such that the user does not want the criterion to deteriorate below the level. The aspiration level is defined as such that the user does not demand the criterion beyond the level.
The user can change $\bar{\bar{\mathfrak{q}}}$, $\bar{\mathfrak{q}}$ and the system solves the following optimization problem:

$$\underset{C,D,\omega,\gamma}{\text{maximize}}\, s_{\bar{\bar{\mathfrak{q}}},\bar{\mathfrak{q}}}(\mathfrak{q}(C,D,\omega,\gamma)), \tag{4}$$

finding the settings C, D, ω, γ.

3.5.1. The Case of a Few Modules

When there are only a few modules, problem (4) can be solved directly by a parametric experiment. Taking various combinations of the values of C, D, ω, γ, one can examine the resulting values of $\mathfrak{q}_1$ i $\mathfrak{q}_2$ by a direct simulation of the work of the modules and based on this one can calculate the values of the scalarizing function $s_{\bar{\bar{\mathfrak{q}}},\bar{\mathfrak{q}}}$, eventually choosing the combination of C, D, ω, γ that gave the biggest $s_{\bar{\bar{\mathfrak{q}}},\bar{\mathfrak{q}}}$.

3.5.2. The Case of Numerous Modules

The number of combinations of the values of C, D, ω, γ grows exponentially with the number of modules under a given sampling density. If there are more than several modules, the computations become unrealistic. Then, however, we can try to compute $\mathfrak{q}$ with statistical methods, using the central limit theorem.
For this we must assume that mistakes of particular modules are independent events. This assumption, a bit disputable, can be substantiated with the difference of the philosophies of the modules.
We shall consider two cases.

Some test example corresponds to a failure. We shall calculate the probability of misclassifying the example by the correlator (i.e., by the whole system).
We treat a_i as independent, discrete probabilistic variables, where $a_i = -1$ with probability $(1 - p''_i)$, and $a_i = 1$ with probability p''_i. We have denoted $p''_i = \varkappa_{i,2}(\vartheta_i)$. Recall that ϑ_i is a parameter indexing the set of efficient results for the ith module. Later ϑ_i will be made variables for each i – they will become decision variables in an optimization task that will serve for tuning the correlator.
We have

$$\mathrm{E}a_i = 2p''_i - 1 \quad \text{and} \quad \mathrm{Var}a_i = 4(p''_i - {p''}^2_i).$$

So for the probabilistic variable $\omega^\top a$ (i.e., for $\sum_i(\omega \cdot a_i)$) we have

$$\mathrm{E}(\omega^\top a) = \sum_i \omega_i(2p''_i - 1) \quad \text{and} \quad \mathrm{Var}(\omega^\top a) = 4\sum_i \omega_i^2(p''_i - {p''}^2_i).$$

We assume that the probabilistic variable $\omega^\top a$, being a sum of many independent variables has the distribution of[2]

$$N\left(\sum_i \omega_i(2p''_i - 1), \sqrt{\sum_i 4\omega_i(p''_i - {p''}^2_i)}\right),$$

where $N(\varepsilon, \sigma)$ denoted the normal distribution with expected value ε and variance σ^2.
The probability of overlooking the failure by the system is

$$\begin{aligned} P(\omega^\top a\gamma > 0) &= P(\omega^\top a > -\gamma) \\ &= 1 - D\left(\sum_i(\omega_i(2p''_i - 1)), \sqrt{\sum_i 4(\omega_i(p''_i - {p''}^2_i))}\right)(-\gamma) \\ &= 1 - D(0,1)\left(\frac{-\gamma - \sum_i(\omega_i(2p''_i - 1))}{4\sum_i(\omega_i^2(p''_i - {p''}^2_i))}\right), \end{aligned} \tag{5}$$

where $D(\varepsilon, \sigma)$ denotes the cumulative distribution function (CDF) of the normal distribution with expected value ε and variance σ^2.

Some test example corresponds to a correct network state. We put $p'_i = \varkappa_{i,1}(\vartheta_i)$. Using a similar reasoning as

[2]Since the variables have different variances, one should assure that none of the variances dominates the others, so that the conditions of the Linderberg-Feller theorem are satisfied. At least, the modules should be tuned similarly, in the sense that they yield misclassification rates of a similar rank.

above, we can calculate the probability of misclassifying this example by the system:

$$P(\omega^\top a\gamma \leq 0) = P(\omega^\top a \leq -\gamma)$$
$$= D\left(-\sum_i(\omega_i(2p'_i - 1)), \sqrt{\sum_i 4(\omega_i(p'_i - p'^2_i))}\right)(-\gamma)$$
$$= D(0,1)\left(\frac{-\gamma + \sum_i(\omega_i(2p'_i - 1))}{4\sum_i(\omega_i^2(p'_i - p'^2_i))}\right). \qquad (6)$$

Finally, in order to find the optimal tuning of ϑ, γ, ω we solve the following optimization problem:

$$\underset{\vartheta,\omega,\gamma}{\text{maximize}}\; s_{\bar{\bar{\mathfrak{q}}},\bar{\mathfrak{q}}}(\mathfrak{q}_1(t,\omega,\gamma), \mathfrak{q}_2(t,\omega,\gamma)),$$

where $\mathfrak{q}_1(\vartheta,\omega,\gamma)$ is given by (6) and $\mathfrak{q}_2(t,\omega,\gamma)$ – by (5). This problem is nondifferentiable due to the nondifferentiability of the scalarizing function $s_{\bar{\bar{\mathfrak{q}}},\bar{\mathfrak{q}}}$ and possible nondifferentiability of the necessary representation of $\varkappa_i$. It seems, however, optimistic that it has as quasi-analytical form, i.e., to compute the value of the goal function for a given argument one does not need to run preprocessors neither use the historical examples. Finding an effective technique of solving the problem is the subject of further research. It may be helpful that functions $\varkappa_i(\vartheta_i)_j$ and $D(0,1)(\cdot)$ are monotone and weights w_i can be assumed positive (if the modules are reasonable their votes should be taken with positive weights).

4. Discussion of the Proposition Soundness and Conclusions

Implementing and validating the proposition given in this paper is a large undertaking, involving an implementation of several of tens of preprocessors and also organizing the supervised learning, solving quite hard optimization problems, etc. Thus such undertaking is a subject of the further research and here we shall only give some partial arguments validating our approach.

The first, fundamental question is whether it is reasonable to combine particular methods from the literature (which can be very complex and subtle) with the quite rough tool of weighted summing. A very interesting example regarding the runtime path method with PCFGs, supports such combining. We shall show that by weighted summing of the "probabilities of the words" we can extend the expressiveness of the method using PCFGs by context handling!

That PCFG are context-free can be expressed as follows: the probability of a parsing subtree depends neither on earlier symbols nor on later symbols (this is a disadvantage of PCGGs, since many events in reality depend on the context [2]). Let us come back to Example 1 from Section 2. The probability of the one-node subtree $Y \to c$ equals to 0.5 independent of which the 1st symbol in the word is. So the occurrence of symbol c is independent of what has happened before (i.e., whether there was a or b in the first position) and amounts to 0.5. In other words, events "the 1st symbol in the word is a" and "the second symbol of the word is b" are independent, which can be written as follows:

$$P_{G_1}(ac) = P_{G_1}(a\star) \cdot P_{G_1}(\star c),$$

where $\star$ denotes any symbol allowed by the grammar at the given position.

Let us define grammar G_2, very similar to G_1 from Example 1 in Section 2 (even identical with G_1 in structure): $\mathfrak{N} = \{S, X, Y\}$, $\mathfrak{T} = \{a, b, c, d\}$, $\mathfrak{R} = \{$
$S \to XY \;\; (P = 1)$
$X \to a \;\; (P = 0)$
$X \to b \;\; (P = 1)$
$Y \to c \;\; (P = 0)$
$Z \to d \;\; (P = 1)$
$\}$

Certainly, for G_2 there also holds the independence of the relevant events:

$$P_{G_2}(ac) = P_{G_2}(a\star) \cdot P_{G_2}(\star c).$$

However, under mixed probability, e.g., $P(.) \equiv 0.5P_{G_1} + 0.5P_{G_2}$, events "$a\star$" and "$\star c$" are no more independent. Namely, we have

$$P(a\star) = 0.5P_{G_1}(a\star) + 0.5P_{G_2}(a\star) = 0.5 \cdot 0.2 + 0.5 \cdot 0 = 0.1,$$

$$P(\star c) = 0.5P_{G_1}(\star c) + 0.5P_{G_2}(\star c) = 0.5 \cdot 0.5 + 0.5 \cdot 0 = 0.25,$$

$$P(ac) = 0.5P_{G_1}(ac) + 0.5P_{G_2}(ac) = 0.5 \cdot 0.1 + 0.5 \cdot 0 = 0.5$$

and finally:

$$P(ac) \neq P(a\star) \cdot P(\star c).$$

Mixing the probabilities introduced the desired contextual information handling to our tool.

Some preliminary experiments with teaching modules have been also performed (see [12] for details). The main outcome is an assessment of the form of set Q, giving flavor of what trade-offs between overlooking failures and raising false alarms can be obtained.

An experimental module had to detect failures consisting in breaking one of the arc of the skeleton computer network of the National Institute of Telecommunications. The module had very limited information about the current network state: as a single example, it had only a sequence of traffic intensities in some other arc at 20 consecutive time moments. To make the job of the module more difficult, the sequence was normalized so as its variance was drawn to 1 and its expected value was drawn to 0 for each example. So the module could only analyze the most subtle properties of the 20-element time sequences. It did it using a simple auto-regressive filter of rank 4 as the preprocessor.

The obtained set Q is shown in Fig. 4. For comparison, a line representing the behavior of the *random classifier* was built in the figure. The random classifier classifies each testing example as positive with probability p or as negative – with probability $1-p$ (independently of the real

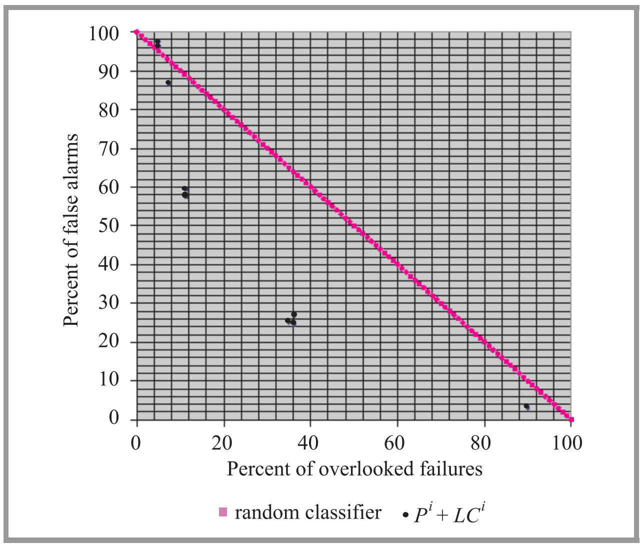

Fig. 4. Attainable results for the AR module.

current network state). By varying p we obtain the whole line in the graph.

Even under a difficult task posing, the AR module exhibited an efficiency clearly better than the random classifier. Correlating several tens of modules of a similar quality would probably be effective.

In conclusion, let us state that it is conceptually possible to join the efforts of various literature detection methods, of which no one is perfect. The main idea of the tool, combining even sophisticated detection methods known from the literature with the mere linear classification seems to be useful in some cases. The most important matters of the further research seem to be: solving the resulting optimization problems, incorporating some at least very rough classification of failures, making a thorough experimental validation.

References

[1] I. Katzela and M. Schwartz, "Schemes for fault identification in communication networks", *IEEE/ACM Trans. Netw.*, vol. 3, pp. 753–764, 1995.

[2] M. Chen, A. Accardi, E. Kcman, J. Lloyd, D. Patterson, A. Fox, and E. Brewer, "Path-based failure and evolution management", in *Proc. First Symp. Netw. Syst. Des. Implem. NSDI*, San Francisco, USA, 2004.

[3] J. Granat, W. Traczyk, C. Głowiński, J. Pietrzykowski, P. Białoń, and P. Celej, "Eksploracja i analiza danych pozyskiwanych z obiektów sieci teleinformatycznej dla wspomagania zarządzania i podejmowania decyzji". Report 06.30.001.1, National Institute of Telecommunications, Warsaw, Dec. 2001 (in Polish).

[4] M. Thottan and J. Chuanyi, "Anomaly detection in IP networks", *IEEE Trans. Sig. Proces.*, vol. 51, no. 8, pp. 2191–2204, 2003.

[5] T. Oates and P. R. Cohen, "Searching for structure in multiple streams of data", in *Proc. Int. Conf. Mach. Learn.*, Bari, Italy, 1996, pp. 346–354.

[6] D. Gürer, I. Khan, and R. Ogier, "An artificial intelligence approach to network fault management", SRI International, 1996.

[7] G. Jakobson and M. D. Weissman, "Alarm correlation – creating multiple network alarms improves telecommunications network surveillance and fault management", *IEEE Network*, pp. 52–59, Nov. 1993.

[8] J. Granat and A. P. Wierzbicki, "Objective classification of empirical probability distributions and the issue of event detection", in *Proc. 23rd IFIP TC 7 Conf. Syst. Modell. Optim.*, Cracow, Poland, 2007.

[9] D. R. Musicant, "Data mining via mathematical programing and machine learning". Ph.D. thesis, University of Wisconsin – Madison, 2000.

[10] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York: Springer, 1995.

[11] P. Białoń, "A linear support vector machine solver for a huge number of training examples", *Contr. Cybern.* (to appear).

[12] P. Białoń, A. P. Wierzbicki, and J. Granat, "Narzędzia monitoringu w zarządzaniu sieciami komputerowymi i bazami danych". Report 06.30.003.7, National Institute of Telecommunications, Warsaw, Dec. 2007 (in Polish).

[13] J. Granat and A. P. Wierzbicki, "Multicriteria analysis in telecommunication", in *Proc. 37th Ann. Hawaii Int. Conf. Syst. Sci. HICSS'04*, Hawaii, USA, 2004, track 3, vol. 3.

Paweł M. Białoń was born in Warsaw, Poland, in 1971. He received his M.Sc. in computer science from the Warsaw University of Technology in 1995. He is with the National Institute of Telecommunications in Warsaw. His research focusses on nonlinear optimization methods and decision support, in particular on projection methods in optimization and support vector machine problems. He has participated in several projects applying decision support in various areas: telecommunications (techno-economic analyses in developing telecommunication networks, network monitoring), also in agricultural and environmental areas.
e-mail: P.Bialon@itl.waw.pl
National Institute of Telecommunications
Szachowa st 1
04-894 Warsaw, Poland