IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 14, 2018, accepted September 30, 2018, date of publication October 8, 2018, date of current version October 31, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2874440

Optimizing Failure Prediction Time Windows Through Genetic Algorithms and Random Forests

JOSÉ MANUEL NAVARRO GONZÁLEZ, JAVIER ANDIÓN JIMÉNEZ, (Member, IEEE), AND JUAN CARLOS DUEÑAS LÓPEZ, (Senior Member, IEEE)

Departamento de Ingeniería de Sistemas Telemáticos, Universidad Politécnica de Madrid, 28040 Madrid, Spain

Corresponding author: José M. Navarro (me@josemnavarro.com)

The work of J. M. Navarro was supported by the Ministerio de Educación, Cultura y Deporte of Spain, under Grant FPU 14/03209.

ABSTRACT Failure prediction is a key component of modern autonomic systems. A crucial decision to take when performing it is which observation window to use, this is, to decide the time period in the past that will be taken into account in order to accurately predict. Currently, this decision is a highly manual process, dependent on expert knowledge. To alleviate this problem, we propose the usage of a customized genetic algorithm alongside a machine learning technique, random forests, which optimizes a novel, multiple observation window schemes that allows for more modeling complexity than other schemes present on the literature. We validate it using ten different events extracted from two real, industrial data sets: one from a high performance computing environment and one from a computer network. We show that our algorithm creates models that optimize performance while reducing the observed time automatically with minimal user input required.

INDEX TERMS Genetic algorithm, machine learning, failure prediction, observation window, random forests.

I. INTRODUCTION

Automating the management of failures in large scale complex systems is of tantamount importance nowadays. Failure prediction models have several parameters to them, specified in [1] as: Δt_l, the lead time given for a prediction (which must be higher than the minimal warning time, Δt_w), Δt_p, the validity period for the prediction, and Δt_d, the data observation window that is utilized to perform the prediction. This model is shown graphically on Fig. 1. For the sake of simplicity, we will refer to Δt_l as l or "lead time" and Δt_d as w, "observation window" or, simply, "window". Regarding Δt_p, we follow the advice found on [2], and employ event-based prediction: we predict whether the failure will occur right after the lead time, which means that, effectively, $\Delta t_p = 0$. This leaves two parameters undefined: How are w and l chosen? It is clear that this decision is crucial in order to correctly predict failures. Of course, it cannot be assigned arbitrarily, as there would be no guarantee of its quality. It wouldn't be wise, neither, to overestimate it and use an exceedingly large value. As w dictates how much time (and, correspondingly, how many events) will be studied and processed, larger values of it lead to higher storage and computation costs. It must be decided by an expert, then. This is a problem: it goes against everything that automatic management efforts seek and incurs in monetary costs and risks. Human expertise is hard to build and not easy to apply to other environments. But, not only that, there are more problems associated to leaving this task to humans: an expert won't be able to design complex observation windows. There is no reason to assume that the optimal time frame to study in order to perform failure prediction is a single one. For instance, failures on the same node may generate other failures almost instantly, while failures on different network nodes may take some time to propagate. And, while it is true that a single, large enough, window would capture all the necessary information, it would also lead to the same problem we talked about with the overestimation situation: storage and computation costs. The optimal solution would be to just take into account the time periods that actually influence future failures and discard the ones that don't.

II. RELATED WORK

While deciding the observation window length seems to be extremely important for Failure Prediction, it is still an open problem with no definitive, general solution. We studied over 200 recent research papers and found more than 20 in which

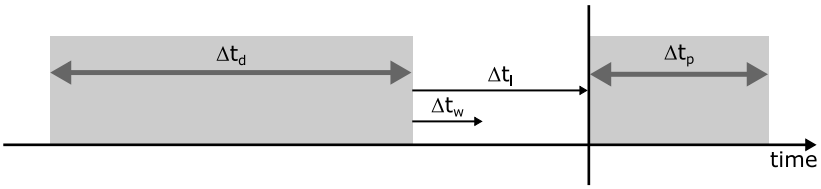


FIGURE 1. Classical Failure Prediction model parameters, as detailed on [1].

the observation window issue is present, pertaining to the following disciplines:

A. SUPERCOMPUTERS AND HIGH-SCALE CLUSTERS

Overall, this area was the most frequent. As we will see in further sections, we observed varied coverage of the window problem. The only papers we found that directly addressed it regarding Failure Prediction were [3], where different observation windows (1, 5 and 12 hours) are tested in order to predict failures on cluster systems, [2], in which the authors compare period-based prediction vs event-based prediction and study the influence of several observation windows and lead times on both, and [4], where the effect of the observation window size in the uncertainty of failure predictions is studied. In contrast, the framework proposed on [5] employs a fixed window based on the user's knowledge. A similar issue is present on [6], where the usage of a Dynamic Bayesian Network is proposed for reliability analysis on service oriented systems but only use a fixed observation window of 20 seconds. The observation window problem arises in several other works in this area, though not applied to Failure Prediction: [7] mentions how assuming a fixed window is not convenient for performing Root Cause Analysis (RCA) and [8] studies the problem of learning patterns in failure logs of distributed systems, in which the time between failures is a key parameter to tune. We consider these issues to be close to our own problem. Lastly, there is another research [9] that proposes a solution for a problem analogous to ours, in the field of log filtering and aggregation.

B. COMPUTER NETWORKS

Networks are also popular when it comes to predicting failures on them. Zhong *et al.* [10] compare four different methods (RIPPER, BayesNet, Random Forest and a Weibull distribution) with four different window schemes (90 minutes, 60 minutes, 80 minutes in two windows and 30 minutes in three windows). There is a similar paper in which network logs are analyzed in six different chunk sizes [11]. These are papers in which the time problem is studied. Conversely, in [12] a 1 hour fixed window is assumed for detecting network anomalies using Principal Components Analyis (PCA) and in [13] RCA is performed by clustering samples of events without studying the effect of time on it.

C. HARD DRIVES AND OTHER AREAS

A similar discipline in which this problem is also present is predicting failures in hard drives. Though we found only three research articles about it, all three assess the observation window's effect on their methods' performance: [14] tests Neural Networks and Support Vector Machines in four different observation windows, [15] applies Classification and Regression Trees against an array of six observation windows and [16] proposes the usage of Gaussian Mixture Models, testing it with several samples. Neither in this area nor in previous ones were any general conclusions extracted, which supports the idea that each dataset and system requires specific knowledge about it.

Apart from large-scale systems, networks and hard drives, the observation window problem has also been found on areas as diverse as software systems [17], [18], theoretical analysis of temporal data [19], [20], the internet of things [21] and even sensor networks on railings [22].

For the sake of clarity of exposition, Table 1 contains a summary of all the relevant references we found.

Overall, we extract the following conclusions from our review: the time problem is present in a variety of disciplines and there has not been a general solution proposed nor has the problem been studied profoundly. The papers that take it into account only test their method against several observation window values, with no assurance that the tested values will be optimal or even close to it. In order to fill this void, we propose a solution based on two ideas: the usage of a custom adapted Genetic Algorithm to optimize the observation window and the definition of multiple observation window schemes. Genetic Algorithms are a subset of Evolutionary Algorithms, algorithms that are inspired by biological evolution and the ideas of recombination and mutation, to solve optimization problems. Specifically, Genetic Algorithms are inspired by the idea of natural selection, evaluating a group of different solutions and recombining the fittest ones to iterate and improve the average solution. This group of solutions is called population and it is originally initialized with random values. The simulation of natural selection is performed based on each population's individual score, calculated using an objective function called ''fitness function''. Once the population has been evaluated and selected, the population of the next generation is created by combining a set of selected individuals and applying mutations with certain probabilities. Our proposal, thus, does not require human interaction nor

TABLE 1. State of the art summary.

Reference	Area	Problem addressed?	Failure Prediction?	Failure Prediction method	Relevant insight
[2]	Supercomputing	Yes	Yes	Bayesian Network	Thorough w and l effect analysis
[7]	Supercomputing	Yes	No		Fixed window not appropriate for RCA
[9]	Supercomputing	Yes	No		Adaptive observation window for log filtering
[3]	Cluster systems	Yes	Yes	SVM, HSMM, rule-based model	1, 5, 12 hours observation window tested
[8]	Cluster systems	Yes	No		Time between failures effect on performance
[4]	Cluster systems	Yes	No		observation window size effect on prediction uncertainty
[6]	Cluster systems	No	Yes	Dynamic Bayesian Networks	20 second fixed observation window
[5]	Cluster systems	No	Yes	Custom	Customizable fixed observation window
[10]	Networks	Yes	Yes	RIPPER, BayesNet, Random Forest and Weibull	4 observation window schemes: • 30 min/3 windows. • 80 min/2 windows. • 60 min window. • 90 min window.
[11]	Networks	Yes	Yes	SVM	5, 10, 15, 20, 30 and 60-minute observation windows
[12]	Networks	No	Yes	PCA	1 hour fixed observation window
[13]	Networks	No	No	Clustering	observation window not specified
[14]	Hard Drives	Yes	Yes	Neural Networks, SVM	12, 24, 48 and 96-hour observation window
[15]	Hard Drives	Yes	Yes	CART	12, 24, 48, 96, 168 and 240-hour observation window
[16]	Hard Drives	Yes	Yes	Gaussian Mixture Models	10, 30 and 50 samples
[17]	Software systems	Yes	Yes	SVM	20 different combinations of observation windows tested
[18]	Software systems	No	Yes	Architectural + failure prediction models	Fixed observation window of 2 minutes
[19]	Temporal data	Yes	No	Sliding window	Different window sizes tested
[20]	Temporal data				Survey: the time problem appears
[21]	Internet of Things	No	Yes	Conditional probability analysis and frequent pattern mining	It is a proposal. No observation window is specified.
[22]	Railing sensor networks	No	Yes	Custom SVM	7 or 14 days fixed observation window

previous knowledge of the system. Additionally, it automatically finds a good trade-off point between prediction performance and observation window size. We chose to develop our solution using a Genetic Algorithm because they are easy to adapt between problems, scale well and are robust and have proven their efficacy in a myriad of environments, such as nanotechnology, plasma physics or bioinformatics. Of course, they are not without hindrances: they may not reach a global optimum and there is no clear process to tune them. We don't consider them serious problems, though, as the former problem can be solved by running them several times and the latter only affects computation time, not its performance. While we did not test them for this work, it would be feasible to use other optimization algorithms, too, and we would like to compare their performance on a future research line.

The usage of evolutionary algorithms in conjunction to a Machine Learning model is a well tested technique with a history of successful applications in several areas, such as electricity price prediction [23] and consumption [24], [25], software reliability [26], financial [27] and stock data [28]. Every work we've found on this area follows the same idea: an evolutionary algorithm is used to optimize the parameters of complex Machine Learning models, such as Support Vector Machines [29] or Artificial Neural Networks [30]. These models have large parameter search spaces and their performance is highly dependent on their fine tuning. This task would, normally, require expertise on the domain area of the data or intensive brute force search, as the performance function is not something that can be easily optimized analytically. Evolutionary algorithms bridge this gap.

Contrary to the trend we discuss on the previous paragraph, our proposal does not employ a Machine Learning algorithm that needs to be fine tuned, we do so because the complexity of our problem is on the data preprocessing: the evolutionary algorithm needs to be able to optimize tens of variables and, for each of those, its performance is interdependent on the values of the others. This makes the optimization problem quite complicated. More details on this can be found on the next section. Considering this fact, we chose to use Random Forests, as it is an algorithm that does not need to be extremely fine tuned to the problem for it to perform well enough. It is an ensemble method (models composed of a group of simpler models) built using Decision Trees [31]. In order to introduce variability in the growing of the trees, it applies randomness in two ways: first, it randomizes the set of variables each tree is grown with. And, second, it randomizes the set of observations it feeds to each tree. Thus, each three is grown seeing a slightly different dataset. This, combined with a large enough number of trees, allows each tree to compensate the shortcomings of its peers. The exact hyperparameters of the model depend on the specific implementation used, but, normally, there are two: the number of trees to grow and how to split each tree [31]. Nevertheless, they are very easy to tune and don't tend to affect performance greatly. This allows the Genetic Algorithm to focus on optimizing the multiple window scheme we propose. There is also another difference between previous works on this discipline and ours: the application area. These techniques haven't been applied to the Network Failure Prediction problem and, as we studied earlier, our proposal is the first effort towards obtaining a general solution for it. Lastly, while a multiple window scheme has appeared in literature before [26], the optimization problem that paper solves is just finding the optimal value of a single number. Our proposal deals with highly complex problems of, potentially, tens of interdependent variables. Such a problem has not been dealt with in related previous works.

We can, thus, sum up the contribution of our paper as a proposal of a general solution of the observation window size problem using a Genetic Algorithm and Random Forests, through:

1) An adaptation of the basic Genetic Algorithm to the problem at hand with heavy customization of several parts of it.
2) The expansion and study of the multi-window scheme that appears in [10] to allow solutions to adapt to complex problems.
3) The validation of our proposal using two real datasets, from two different areas, against other window configuration methods inspired by previous literature.

III. PROPOSAL

Based on how it is treated on previous researches, the window length problem for Failure Prediction can be formalized as:

$$\max_{w} P(w) \tag{1}$$

where $P(w)$ is a performance function that indicates how well a model is predicting failures and w is the observation window that is taken into account by the model. As we have already discussed, this model is overly simplistic and can be improved in two fronts:

1) Costs: a crucial issue in terms of model scalability (in situations where, for example, hundreds of events are to be predicted), limiting the amount of data that is utilized reduces computation and storage costs. While it may be possible to argue that Big Data solutions render this limitations invalid, storing and using too much data is not without its hindrances: there is the risk of introducing random noise into the model that may lead it to find spurious relations, which would harm its generalization capabilities when deployed on a production environment.
2) The second area in which the original formulation can be improved is in terms of the actual window, w. We propose to change w into $W = \{l, w_1, d_1, \ldots, d_{n-1}, w_n\}$ where l is the lead time, the time between the moment when the prediction is done and the forecasted moment of failure (in most papers, this is also taken as a fixed value), w_i are different observation windows and d_i are time delays between windows: time periods

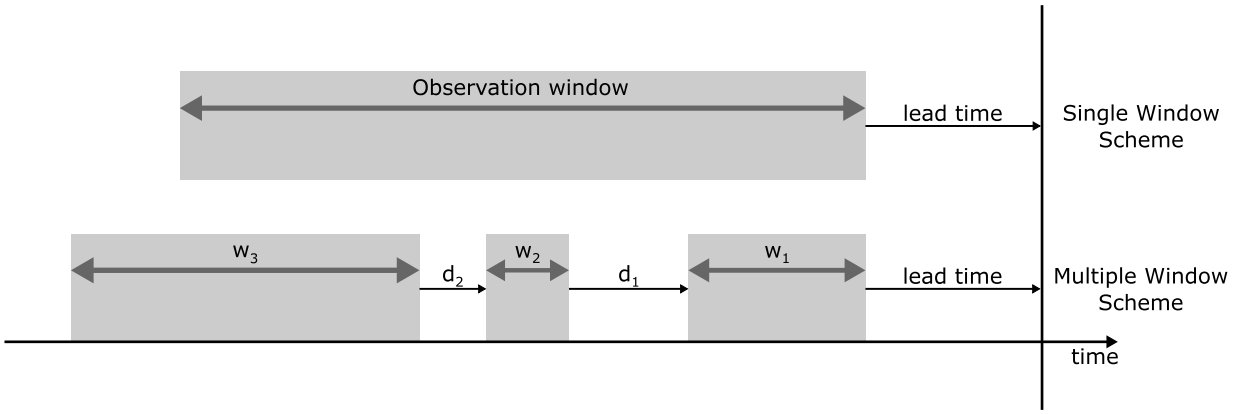


FIGURE 2. Single window scheme versus complex window scheme examples.

that the model does not use on its calculations that allow windows to be separated in time. We introduce this concept in order to allow the model to capture different relevant intervals without including any unuseful period between them: this alleviates storage costs and avoids including non important data or noise to the model. As a way of illustrating this concept, Fig. 2 shows the classical observation window versus an example of an observation window scheme that could be generated using our proposal.

Adding these two improvements together, the updated formulation of the problem would be:

$$\begin{cases} \max_W P(W) \\ \min_{\{w_1,\ldots,w_n\}} \sum_{i=1}^{i=n} w_i \end{cases} \tag{2}$$

While that is the actual mathematical expression of the problem, we will add a way of deciding the importance of both functions on the model, to allow for further customization of our proposal to let it adapt to different scenarios. Additionally, we must make one remark on this specific problem formulation regarding the lead time. Ideally, it's best to have a lead time as high as possible, so it would be sensible to include it as another condition to maximize on the problem definition. We have opted, though, to establish a minimum value for it (that should be decided taking into account the system and the time needed to fix it), in order to avoid additional restrictions to the models' performance. If having a high lead time is a crucial element of the system, adding the additional constraint is trivial and the algorithm should also perform correctly under it.

We will now introduce the core parts of our proposal. Though the basic philosophy and purpose of our proposed Genetic Algorithm is the standard one, to find optimal values of sets of variables, there are several parts of it we custom tailored to the prediction problem. Fig. 3 shows a general outline of the steps of our algorithm and which steps have been tailored to our problem. First, we will define the variables to optimize (initial population generation on Fig. 3): six observation windows, w_i, five time delays, d_j, where $i = 1, \ldots, 6$ and $j = 1, \ldots, 5$ and a lead time (we employed six windows as a maximum bound for the allowed window scheme complexity; this number can be increased at the cost of an increased computation time).

For its initial population, each individual is composed of twelve numeric variables, corresponding to $l, w_1, d_1, \ldots, d_5, w_6$. We give them values as follows: for the lead time, we use a Poisson distribution of $\lambda = 3$ minutes with a minimum of 1 minute, a design choice that should be taken based on the amount of time needed to prepare for a failure on a specific system. The usage of this distribution is based on the following reasoning: while we assume that most failures will be close in time (which is what a low λ value generates), we want to allow the algorithm the possibility of exploring other, unexpected, solutions with higher lead times. Any other distribution that mostly generates low values with sporadic high ones would be a proper candidate for this too. Overall, the selection of the distribution should not be something that makes or breaks the general algorithm, but it could cause it to converge in more time.

The second value to initialize is each window's length: for each of them, their initial values are $M_{15} + \eta$, where M_{15} is the median value of time difference between 15 events on the dataset and η is a value extracted from a normal distribution with 0 mean and $M_{15}/5$ standard deviation (our tests showed 15 events and the specified deviation rates to be large enough for the algorithm to work properly. We did not include that search for the sake of brevity. If needed, these values could be found by performing a grid search, for example). The idea behind this value is to supply the algorithm with large initial values, as, as will be shown, it will tend to shrink the windows. Thus, we expect it to start on a sufficiently large value and diminish the windows as much as possible while optimizing the model's performance. While it may seem that this task is a matter of just shrinking the window, which could be a task easily done by simpler algorithms, such as binary search or hill climbing, it is not as simple as that. First, we are optimizing multiple variables at the same time, which renders binary search unusable. Furthermore, we assume the

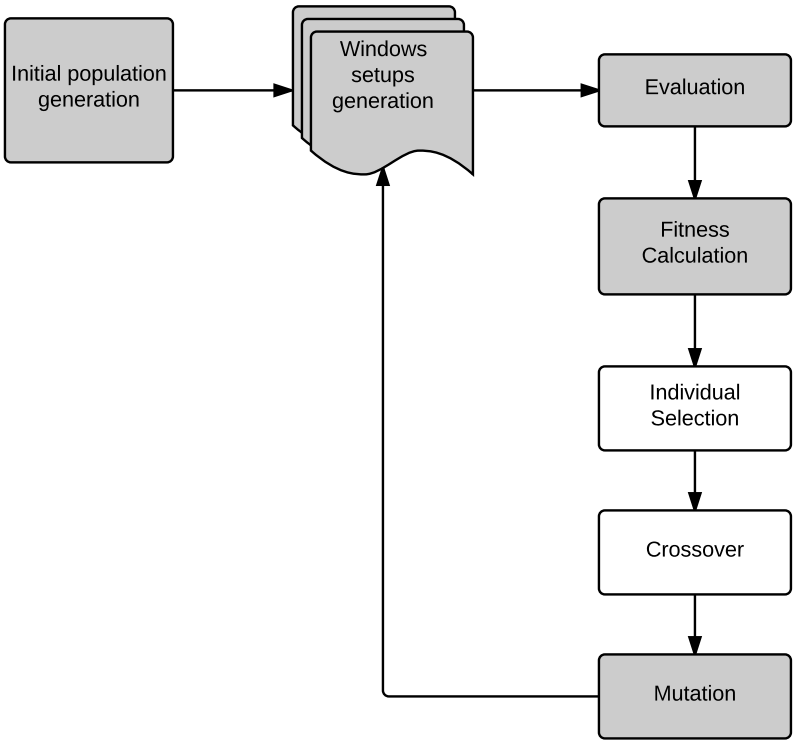


FIGURE 3. Genetic algorithm steps. Grayed out boxes indicate customized steps.

objective function will contain local minima, making the use of hill climbing algorithms ineffective. While this is an assumption, we err on the side of precaution, as a Genetic Algorithm will work whether it is true or not, and a hill climbing algorithm would not work if local minima exist. Growing a small window would also work, but it would need more time to converge and additional fitness and mutation criteria to avoid the windows to grow uncontrolled. Overall, it is more aligned with our objectives to shrink large windows.

The initial population created by the process described on the previous paragraph is, then, supplied to the algorithm, which, alongside the original event sequence, generates the window observations that constitute the input of the Machine Learning model (windows setup generation on Fig. 3).

Our Genetic Algorithm follows the standard phases: population evaluation, individual selection, cross-over and mutation. There are several specific decisions or design criteria about them we would like to discuss. The population evaluation phase is divided in two substeps:

1) Evaluation: as each individual is a set of observation window lengths and time delays, for each individual we first process the sequential data according to them, creating the aforementioned windows, and then train and evaluate a Machine Learning model using the processed data. In that way, each individual shapes how the data are preprocessed before feeding it to a (identical for everyone) training model. For our experiments, we chose to use Random Forests [32], as they are very easy to tune and show very good general performance.

We evaluated our models using the Area Under the Roc Curve (AUC) [33], which can be calculated as follows: for a binary classifier such as ours, models will output a certain number, p (extracted from a continuous random variable P), that is compared with a threshold, t, in order to decide whether to classify a sample as positive ($p > t$) or negative ($p \leq t$). Given these values, p follows a certain probability density function, $d_p(p)$, when a sample is actually positive, and another probability density function, $d_n(p)$, if the sample is actually negative. This allows us to calculate the true positive rate, TPR, and the true negative rate, TNR, as:

$$TPR(T) = \int_T^\infty d_p(p)dp \tag{3}$$

$$TNR(T) = \int_\infty^T d_n(p)dp \tag{4}$$

which allows us to define the AUC as

$$AUC = \int_{-\infty}^{\infty} TPR(T)TNR'(T)dT \tag{5}$$

This value, when using standard units, goes from 0 to 1, where 0.5 equals a random model (one that does not take into account the input variables to perform a prediction) and 1 is a perfect model. Additionally, the AUC equals the probability that a model will classify a random positive sample higher than a random negative sample. Lastly, for our experiments, we followed a 3-folds cross validation scheme [34]. While it is lower

than the usually recommended 5 or 10-folds, we had to settle on a tradeoff between accuracy and computation time. If accuracy is an extremely important criterion, the number of folds could be increased. Furthermore, as each configuration is reevaluated on each generation, the impact of this accuracy loss is decreased.

2) Fitness calculation: the AUC score obtained on the previous section would be a suitable way of calculating each individual's fitness. But the prediction scenario presents an issue that should be considered: collecting and processing data has an associated cost. When dealing with large scale systems, storing data recklessly can quickly turn into an unmanageable situation. Thus, we defined a fitness metric that penalizes large windows using a Window Shrinkage Factor (WSF). It is defined as:

$$F = \alpha * AUC + (1 - \alpha) * WSF \tag{6}$$

where

$$WSF = \frac{I_w - G_m}{G_M - G_m + 1} \tag{7}$$

being I_w an individual's window length and G_M and G_m the generation's maximum and minimum window lengths, respectively. While we could have formulated the optimization problem as a bi-objective one, we settled on the current formulation, as α allows the end user to customize the algorithm to their specific problem at hand. This metric is reminiscent of the regularization methods for training machine learning models, whereas instead of shrinking a model's variables, our Window Shrinkage Factor (WSF) makes the windows grow smaller. Thus, depending on the costs of collecting data, the user can tune how much performance he wants to sacrifice in trade of smaller windows. We perform a study of this parameter's values on the next section. Additionally, common multivariable optimization techniques, such as grid searches or univariate searches, would not be effective, as there is a strong interaction between the both variables we optimize: model performance is highly dependent on the total window size.

3) For the selection and crossover steps we employed standard procedures: tournament selection and single-point cross-over. The mutation part, though, is also heavily customized: for each individual, there is a 5% chance that its parameters are completely randomized. If not, for each feature, there is a 10% chance of it to be mutated in the following way:

$$\hat{x} = x + (\mu - 0.3) * M_{15}/10 \tag{8}$$

where $\hat{x}$ is the feature's new value, x is the feature's previous value, μ is a random value extracted from a normal distribution of 0 mean and 1 standard deviation. The addition of the second member of the equation, 0.3, generates a random value that will be: centered on a negative value and will have a magnitude around one tenth of the median separation between events. What this does is to, on average, shrink the windows by sensible steps, contributing to creating small windows. If the feature is 0, there is a 5% chance of it getting a random value instead. These mutation values and mean shifts were decided ad hoc by us after testing a small range of possible candidates. A standardized method of deciding their values and an analysis of their effect on the algorithm's performance is a line of future work for us.

We tested our algorithm customization against simpler schemes, namely:

1) Standard initialization: initializing each variable with $V = U(0, M_{15})$.
2) Standard mutation: same scheme as our proposal but changing equation 8 to $\hat{x} = x + \mu * M_{15}/10$. This option doesn't progressively shrink the windows.
3) Standard mutation and initialization: initializing each variable with $V = U(0, M_{15})$ and changing equation 8 to $\hat{x} = x + \mu * M_{15}/10$.

We evaluated them running the algorithm for one prediction model, five times for each scheme. Out of the four tests (three simple ones and our proposal), the ones that included standard initialization were clearly worse: the individuals had much more dispersed values and the algorithm did not converge in 100 rounds. The difference between the standard mutation scheme and ours was more subtle: while mutating with just a gaussian distribution achieved slightly higher fitness values, we checked that it was due to the algorithm getting stuck in smaller window sizes. Our proposal was able to move to higher window sizes, getting higher AUC overall and exploring a larger variable search space.

Algorithm 1 Genetic Algorithm Proposal

```
Generate initial population (algorithm 2)
for 100 rounds do
    for each individual j do
        Evaluate Random Forest using individual's
        window scheme
        Calculate each individual's fitness (algorithm 3)
    end for
    Tournament selection of 4 individuals
    Single point crossover with probability 1
    Mutate new individuals (algorithm 4)
end for
```

Everything we've exposed about our proposal is summarized in Algorithms 1 to 4, where α is the fitness balance factor, M_{15} equals the median value of time difference between 15 events in the dataset and each individual in the initial is represented by the following coding scheme: $V_j = \{l_j, w_{j,1}, d_{j,1}, w_{j,2}, d_{j,2}, w_{j,3}, d_{j,3}, w_{j,4}, d_{j,4}, w_{j,5}, d_{j,5}, w_{j,6}\}$, where $j = \{1, \ldots, n\}$, where every variable is a numeric value and n is the population size, for which we utilized 20 in our experiments.

Algorithm 2 Generate Initial Population

for each individual j **do**
  $l_j = poisson(\delta = 3)$, min = 1 minute
  for $i = 1$ **to** $i = 5$ **do**
    $d_{j,i} = M_{15} + gaussian(\text{mean} = 0, \text{sd} = M_{15}/5)$
    $w_{j,i} = M_{15} + gaussian(\text{mean} = 0, \text{sd} = M_{15}/5)$
  end for
  $w_{j,6} = M_{15} + gaussian(\text{mean} = 0, \text{sd} = M_{15}/5)$
end for

Algorithm 3 Calculate Each Individual's Fitness

Require: AUC_j = Each individual's Area Under ROC Curve
Require: α = Fitness Balance Factor
for each individual j **do**
  $sum_j = \sum_{i=1}^{6} w_{j,i}$
end for
$G_m = \min(sum)$
$G_M = \max(sum)$
for each individual j **do**
  $WSF_j = (sum_j - G_m)/(G_M - G_m + 1)$
  $Fitness_j = \alpha * AUC_j + (1 - \alpha) * WSF_j$
end for

Algorithm 4 Mutation

for each individual j **do**
  if $U(1, 100) \leq 5$ **then**
    Randomize individual as described on Algorithm 2
  else
    for each individual's variable, $V_{j,m}$ **do**
      if $V_{j,m} == 0$ **then**
        if $U(1, 100) \leq 5$ **then**
          $V_{j,m} = M_{15} + gaussian(\text{mean} = 0, \text{sd} = M_{15}/5)$
        end if
      else
        if $U(1, 100) \leq 10$ **then**
          $V_{j,m} = V_{j,m} + (gaussian(\text{mean} = 0, \text{sd} = 1) - 0.3) * M_{15}/10$
        end if
      end if
    end for
  end if
end for

IV. VALIDATION

To test our approach, we must choose some other window scheme to evaluate it against. We settled on using a heuristic approach based on a grid search, as it is the only method present on literature. We used fixed values and a method based on the dataset's features to decide on the values used. We combined different lead times (1, 5 and 10 minutes) with different window combinations:

- Single windows: 5, 10, 50 and 100 minutes and M, $3M$, $5M$ and $30M$ seconds.
- Six equal-sized windows of: 5 and 10 minutes and M and $5M$ seconds.

Here, M indicates the median time separation between events on the dataset. We end up with a search body of 3 lead times and 12 window schemes, this is, 36 different experiments. We tested all methods with two different real datasets, pertaining to the two most frequent disciplines where the window problem arose: high performance computing and networking.

A. HIGH PERFORMANCE COMPUTING DATASET

For this area we chose to use the Los Alamos National Laboratory Dataset [35]. It details "cluster node outages, workload logs and error logs" of a system composed of 22 cluster systems, ranging from December 1996 until November 2005. There are 23739 error events on the whole file, with 26 variables giving various details of them. For our analysis, we created a new error type variable, containing the following information: the system in which it happened, the major type of failure (one of either "Facilities", "Hardware", "Human.Error", "Network", "Undetermined" or "Software") and the subtype of error, yielding 727 different error types. As an example, an instance of our error type may look like "(system)3-Hardware-Maintenance", where "Maintenance" is a subtype of Hardware error. We used this information along with when the event happened to create a sequence of events that can be windowed and fed to a Machine Learning algorithm. Out of the 727, we chose five of the most frequent ones to test with our algorithm. They are:

- S1: a CPU problem on system 18.
- S2: a CPU problem on system 19.
- S3: a Memory Dimm problem on system 2.
- S4: a Node Board problem on system 2.
- S5: a Memory Dimm problem on system 18.

B. NETWORKING DATASET

For the networking area there is a lack of good quality, public, error datasets. We resorted to using a private dataset provided to us by a company. It contains network events extracted from a network manager, categorized by when they happened, the device they happen on and the severity of it. It comprises 206 days, with 21442 events happening on that time on the system, with 1628 of those being critical events and 1539 being major ones. It is composed of one router, six switches and fourteen virtual machines. We joined for our error type the event and location, in a similar process to the one we followed for the LANL dataset. This rendered a similar event sequence that can be windowed and processed as before. Again, based on their appearance frequency and importance, we selected five events to use:

- N1: critical severity. Alarms of a software issue on a virtual machine of the network.
- N2: critical severity. Indicates a CPU threshold violation on a virtual machine.

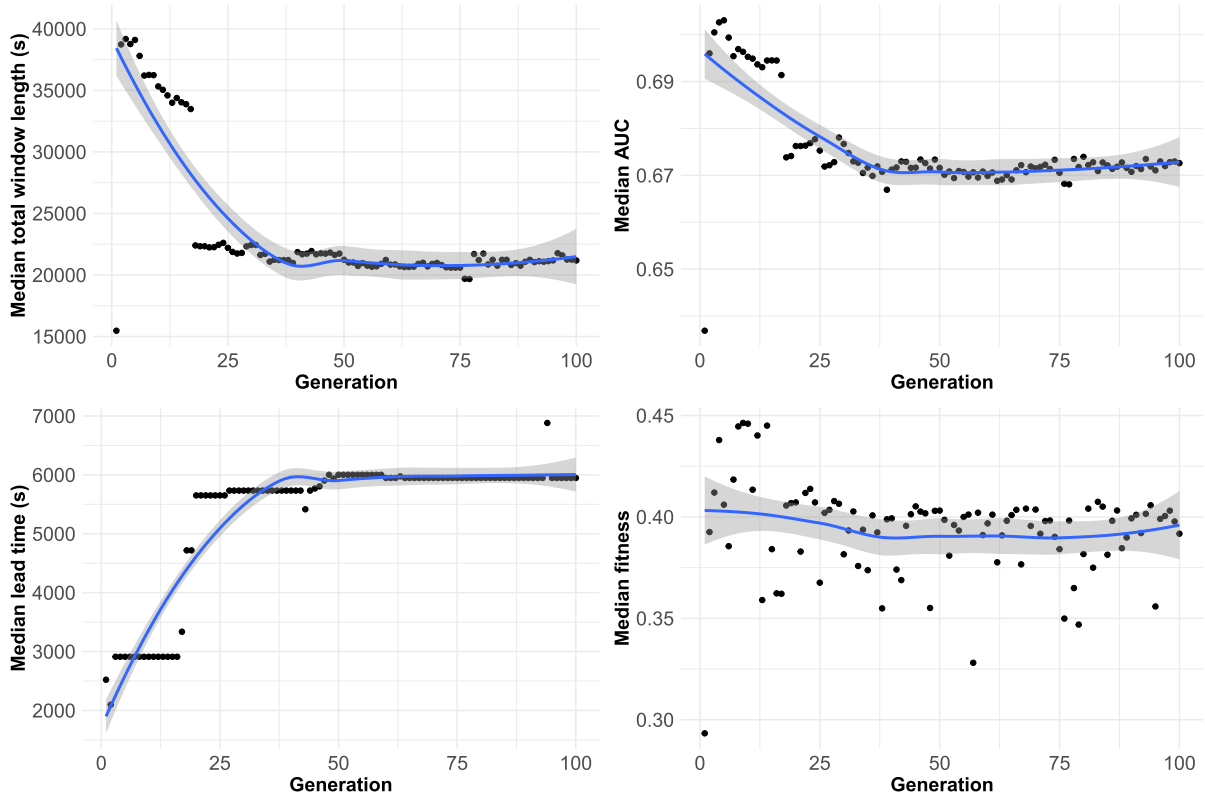


FIGURE 4. Genetic algorithm behavior for event S1.

- N3: major severity. A communication link of a virtual machine is down.
- N4: major severity. The network router is not responding to primary management requests.
- N5: major severity. A process or service has stopped on a virtual machine.

V. EXPERIMENT DETAILS

All experiments were run on a cluster of eight HP ProLiant SL250s Gen8 with the following specifications: two Intel Xeon e52630v2 2.6GHz (6 cores each) and 32GB of RAM. The Genetic Algorithm's parameters were: 20 individuals population, 100 generations, tournament selection and single-point crossover with a probability of 1 (we always performed crossover). We programmed it on Spark 2.2 using the Scala programming language. The Random Forests implementation we used was the one found on Spark's Machine Learning library, SparkML. We trained forests with 50 trees, a maximum tree depth of 10 and we used the gini coefficient for the impurity measures. These are the specific parameters for this Random Forests version. The posterior analyses were performed using the R statistical environment.

VI. EXPERIMENTS RESULTS

We applied our algorithm and the grid search heuristic we described in the previous section to the five selected events from each of our datasets. We perform it once for every event to analyze, as there is no guarantee that they will show similar behavior. In fact, it would not be a surprise if they performed in a completely different way, an assumption that becomes truer as the analyzed system grows more complex. For the Genetic Algorithm part, we first obtained the window combination that yielded the best AUC of the last generation and then tested it against the whole dataset following a 10-folds cross validation scheme, repeated 5 times to reduce the potential variability of the results. Each option of the grid search approach was tested with the same cross validation process.

A. GENETIC ALGORITHM PERFORMANCE

The first thing we should study would be the Genetic Algorithm's population evolution through generations. For it to work properly, we would expect the fitness measure and, correspondingly, the AUC to increase up to a certain point and/or the window length to decrease as much as possible. For this preliminary run, we used a value of 0.9 for α.

1) HIGH PERFORMANCE COMPUTING DATASET

For our first dataset, we detected three different behaviors for the algorithm. For events S1 and S2 (event S1 depicted on Fig. 4, in terms of the median population behavior for total window length, selected lead time, AUC and fitness

TABLE 2. Window schemes selected by the Genetic Algorithm for each event on the High Performance Computing Dataset (measured in seconds).

Event	lead time	w_1	d_1	w_2	d_2	w_3	d_3	w_4	d_4	w_5	d_5	w_6	$\sum_{i=1}^{6} w_i$
S1	5947	14766	0	5275	35	167	0	63	0	0	0	0	20271
S2	11012	16298	0	370	0	0	0	0	0	0	0	0	16668
S3	1656	31	0	18450	0	0	0	0	160	0	370	0	18481
S4	60	377	0	18528	0	3	226	0	0	0	0	0	18905
S5	10854	10530	901	2538	0	14971	0	386	0	0	197	0	28425

score accross the generations), the algorithm starts with very high window length values, which render a high AUC. The shrinkage effect comes into play, and the window converges rapidly to a very much smaller length while only losing a trivial amount of AUC score. The algorithm's performance for events S3 and S4 follows more closely what we expected: it diminishes the window length while achieving an increase in the AUC score. Lastly, event S5 seems to follow a hybrid behavior between the two we discussed: first it starts with an extremely large window value and shrinks it very fast but, instead of then plateauing on performance and increasing the lead time, it manages to start increasing the AUC score again. One potential problem we must address is the fact that, while the AUC or the window length seem to follow a clear trend, the fitness (specially for event S5) is much more erratic. We assume this is because the fitness function is a complex one, that combines the output of the Machine Learning model and the Window Shrinkage Factor. As its behavior does not seem to be negatively affecting the algorithm's performance, we won't address it further.

As another way of checking if our proposal is working correctly, we show on Table 2 the selected window schemes for each event, this is, the individuals with the best AUC of the last generation for each event (the best fitness' individual may also be chosen). We draw several conclusions from it: as it corresponds to a dataset with a low event density (at least, compared to our networking one), the lead time can be large without it affecting the model. This does not happen, though, for event S4. Apart from this small difference, all of the events end up with a similar window configuration: a single large window, normally around 18000 seconds, except for event S5, which employs a 28000-second one. It is remarkable how, for this dataset, the algorithm has selected a single window, not employing the multiple window scheme capabilities it has. This finding is a good sign of the algorithm's expresiveness, as it indicates that it is able to simplify the solution if the complexity of a multiple window scheme is not needed. While there are some cases in which $d_i > 0$ while $w_{i+1} = 0$, these are artifacts of the optimization process, that does not take into account the meaning of each variable. Nevertheless, the optimization process works as expected.

2) NETWORK DATASET

On Fig. 5 we show the Genetic Algorithm's behavior plots for event N2. This time, though, we only show the first 50 generations, as the algorithm converged faster. Overall, it seems to work as expected. The window length greatly decreases, while the AUC increases. Another positive side-effect is that the lead time is also increasing, as it happened on the previous dataset. It seems to be, then, an emergent property of our algorithm. For event N4, the algorithm works along our expectations: the window length is optimized extremely quickly (in less than 20 generations) but the fitness (and, correspondingly, the AUC) is optimized until it reaches the plateau of the local performance optimum. The cause for the ''jumps'' on the fitness plot could be due to the algorithm trying to lower the window too much, causing the AUC to be lowered, correcting it and then trying it to do so again. This behavior could probably be averted by setting an additional stopping criteria to the generation limit but, as it is, it is not hurting the algorithm's performance, only its computation time, which we will discuss further down.

Lastly, Table 3 shows the selected window schemes for each event. At first glance, a clear observation is how different the selected schemes are, a totally different situation from the one we had on the LANL dataset. This is a good sign: even though the initial populations were generated equally for all executions of the algorithm, it has been able to adapt them to fit each specific problem, even if they are wildly different (as the differences between solutions imply). Furthermore, and in contrast to the previous dataset, this time the algorithm has chosen a multiple window scheme for several events (N2, N3, N4 and N5). This shows how there are instances where more complexity is needed to accurately predict the target event. Another interesting fact is how, for event N1, it managed to get a high lead time, as we observed on the behavior plots previously. One point for improvement would be, though, to include a preference for higher lead times on the fitness function. As it is not explicitly included on it, our current algorithm would not select the model with a larger lead time between identical models.

Overall, the algorithm seems to be working as expected and, furthermore, it has shown its adaptation capabilities, as it created large single windows for the LANL dataset and varied, small, multiple windows for the Networking dataset. Another good sign is how, in hindsight, a maximum of six windows has shown to be enough to successfully model all events, as their optimal solutions are below that quantity.

B. GENETIC ALGORITHM STABILITY

It is crucial to ensure that the algorithm is stable between runs, both in terms of performance and in terms of the chosen

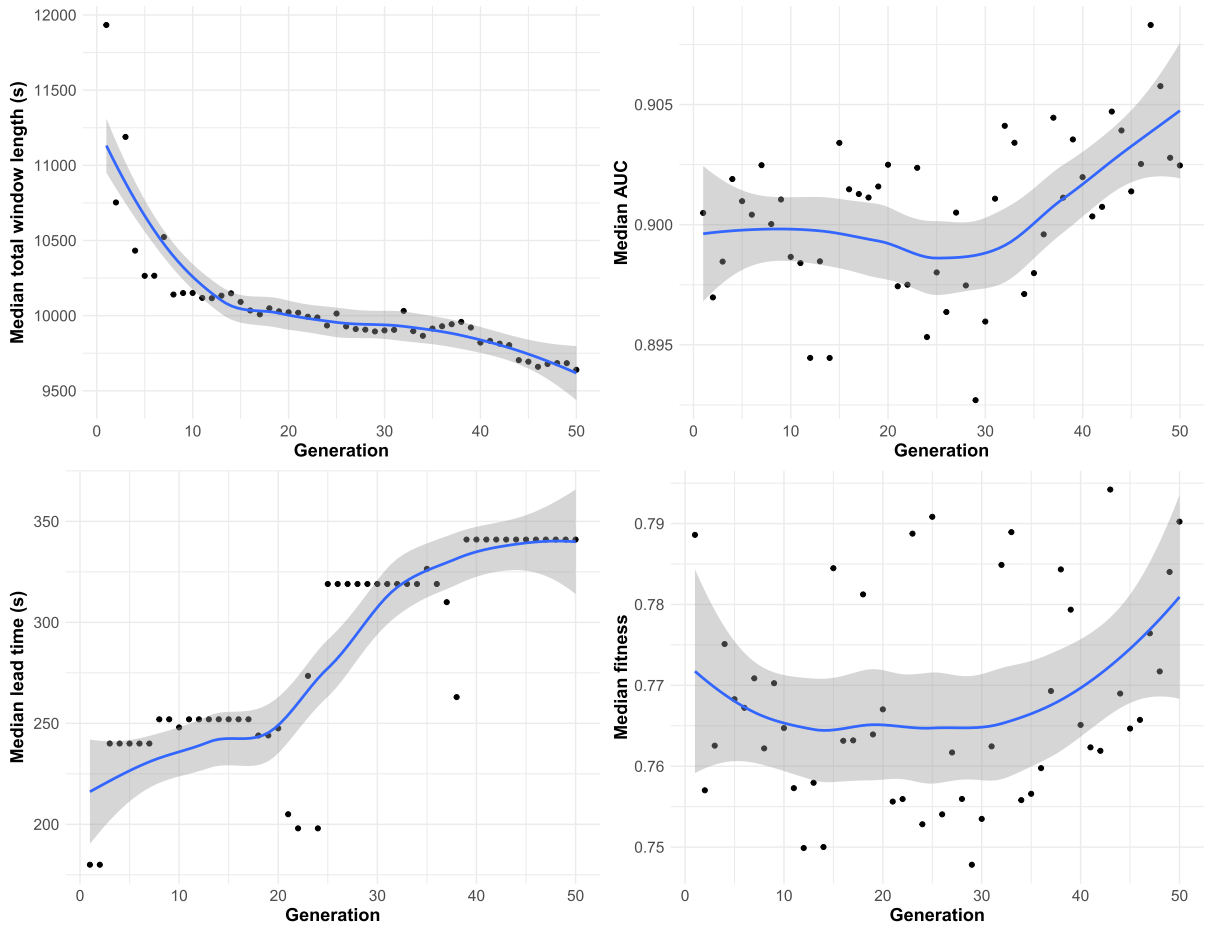


FIGURE 5. Genetic algorithm behavior for event N2.

TABLE 3. Window schemes selected by the Genetic Algorithm for each event on the Networking Dataset (measured in seconds).

Event	lead time	w_1	d_1	w_2	d_2	w_3	d_3	w_4	d_4	w_5	d_5	w_6	$\sum_{i=1}^{6} w_i$
N1	1289	572	0	796	0	705	0	10	65	9	128	0	2092
N2	341	2459	55	1371	159	1415	145	1068	0	1591	269	1728	9632
N3	60	1476	75	2001	68	1321	0	1600	0	702	257	836	7936
N4	60	412	103	330	44	11	0	0	0	0	28	0	753
N5	88	1753	94	1773	235	1669	159	963	32	1276	0	1446	8880

window scheme. While we've checked that the algorithm performs as expected on several events, we've only run it once for each of them. To check its stability, we ran the algorithm ten times for the N4 event. There is one more thing to consider, though: the effect of α, the Window Shrinkage Factor parameter. As it alters the fitness function, it surely affects the algorithm's stability. So we expanded our experiment and ran the algorithm 10 times for $\alpha = \{0.75, 0.8, 0.85, 0.9, 0.95\}$. Fig. 6 shows the density of the median results for the individuals of the last generation of each run. For other values of α (0.75, 0.8 and 0.95) we observe too much dispersion in terms of either the AUC or the window lengths. Both 0.85 and 0.9 seem to be stable enough to be used. We also ran Kolmogorov-Smirnov tests on the obtained performance distributions and confirmed that they can be assumed to follow a normal distribution. So, we consider the algorithm stable in terms of performance for these values.

There is one last area we must study to confirm that the algorithm is stable, though: the specific window configurations it chooses. To test this, we took the 10 runs from the $\alpha = 0.9$ test for the N4 event and did the same for the S1 event. Then, we took the individuals of the last generation of each run and plotted the obtained schemes. These are shown on Fig. 7, where the horizontal axis specifies which time period(s) each individual focused on, whereas the vertical axis shows each individual solution. Thus, it is a graphical representation of the selected windows on the last generation of several runs of the algorithm. N4's results (right part of

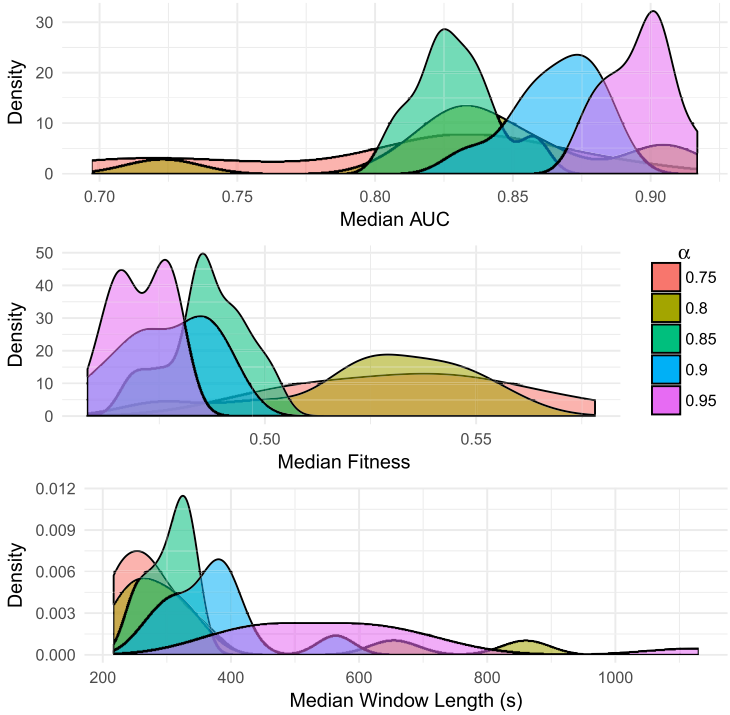


FIGURE 6. Genetic algorithm stability for 10 runs of different values of α in terms of median AUC, median fitness and median window length of the individuals of the last generation.

the figure) are clearer: the algorithm always focuses on the same period (while there are some mutation remnants that focus on different periods, they are a negligible quantity), focused around the first 450 seconds. S1's results are trickier: while it may seem that there are two different behaviors (the first one centered around 0 to 20000 seconds and the second one focused between 17500 and 40000 seconds, approximately), we must consider that the only thing relevant to the algorithm's performance is whether it captures the events that influence the target event's appearance. In this sense, we can observe that both behaviors overlap on a certain time gap, around 17500 to 20000 seconds. This leads us to think that the events relevant to a good model accuracy happen on this time interval. In fact, studying these window configurations would be something very interesting that could lead us to reduce the observation windows even further. This could be a promising line of future work.

Based on the two experiments we've shown, we consider the Genetic Algorithm stability has been proven. In order to compare its results, from now on, we will only run the algorithm once.

C. AUC RESULTS

Having confirmed that the Genetic Algorithm is working as intended, we have to check how it compares against other solutions in terms of model performance. This is, how well the Machine Learning model trained with the selected window scheme performs against other similar models trained with different window schemes. As, regarding storage, what matters is the total combined observation window that will be analyzed, we will assign a total window length to each window scheme and plot the performance results along this axis.

1) HIGH PERFORMANCE COMPUTING DATASET

Fig. 8 shows a summary of the algorithm's performance against the other fixed schemes in function of the total window length. The results were obtained averaging the obtained performances by window size. The variances were negligible. There are, again, three different behaviors of the algorithm, regarding how it compares to the fixed values models performance. Events S1 and S4 show similar results: the algorithm is able to achieve higher performance than similar window values for the fixed approach, being lower, though, that the AUC obtained by the extremely large window options. Events S2 and S5 perform even better: even using relatively small windows compared to the fixed schemes options, the algorithm performs better than all of them, obtaining extremely high AUC scores. The only event in which the algorithm does not outperform the fixed schemes in one way or another is event S3, which appears to be a harder to predict event, for both the fixed schemes and for our algorithm. Another interesting observation that can be made from these graphs is how the multiple window scheme works worse than the single window scheme for this dataset. This is something that was hinted previously by the Genetic Algorithm's

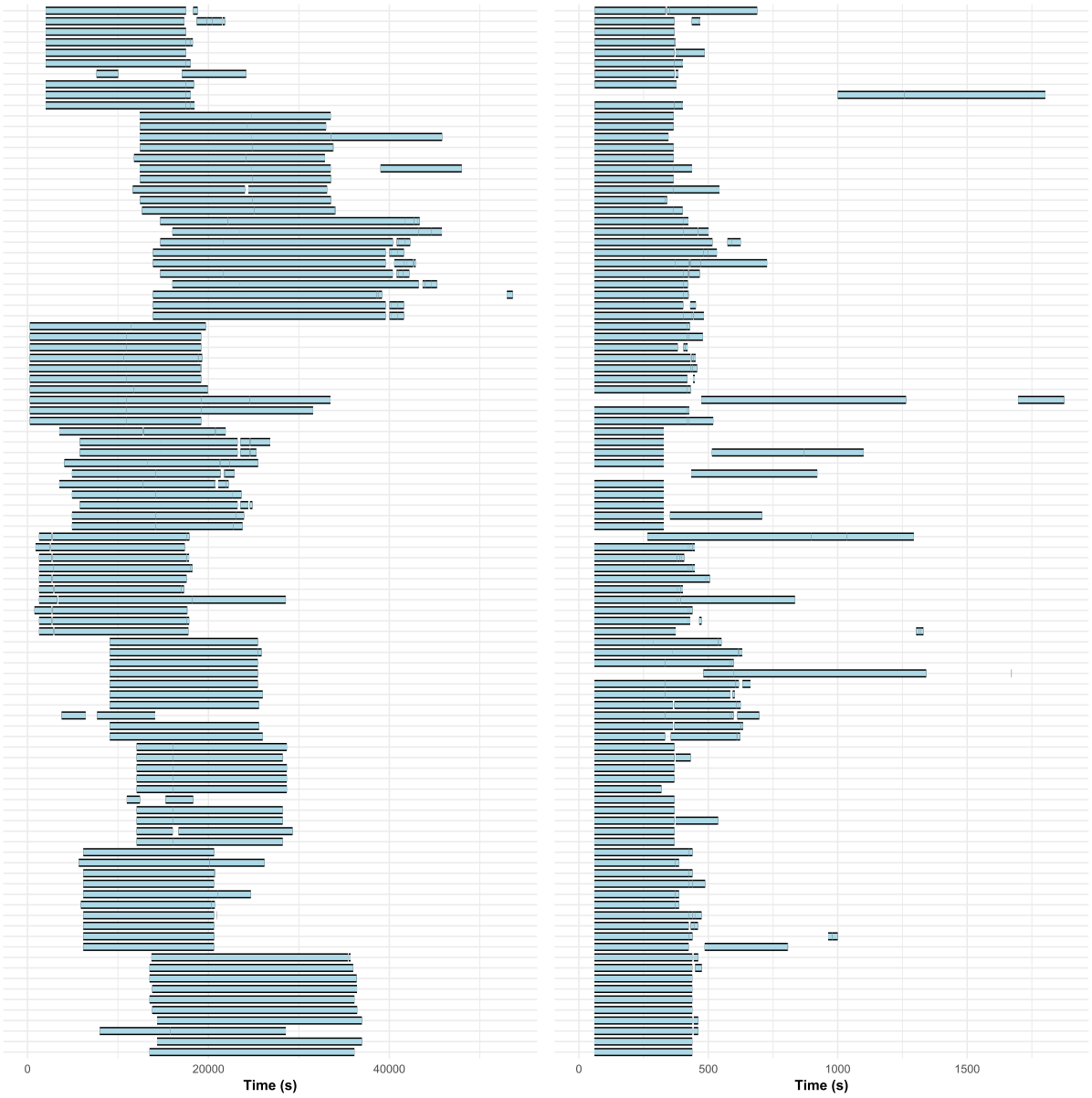


FIGURE 7. Window configurations for the individuals of the last generation of 10 different Genetic Algorithm runs for events S1 (left) and N4 (right).

chosen schemes and is confirmed now. Additionally, in order to compare our proposal's performance against each single other scheme, we took all the cross validation raw values (the AUC obtained on each fold for each model) and performed an ANOVA and a Tukey's range test [36], that runs one on one comparisons between the different schemes and gives an estimation of the difference between values and a p-value to indicate if the difference is significant. We set a significance level, α, of 0.05, applying the Bonferroni correction [37], in order to reduce the Type I error chances. Based on this value, we considered a scheme better than another if the p value was less than the set significance level and the effect size (the difference of median AUC) was positive. For this dataset, the results were good: out of the 36 schemes against we tested our proposal, it was (in average) better than 27 of them and worse than 4 of them, with no difference to five of them. This result is supported by what we just studied: extreme window values yield a higher performance, but our method finds a sensible trade-off between performance and storage costs.

2) NETWORKING DATASET

For the Networking dataset, the chosen schemes were closer in total window length and the obtained variances were not negligible, so we chose to plot them as boxplots. For the sake of clarity, we only chose the best five schemes for

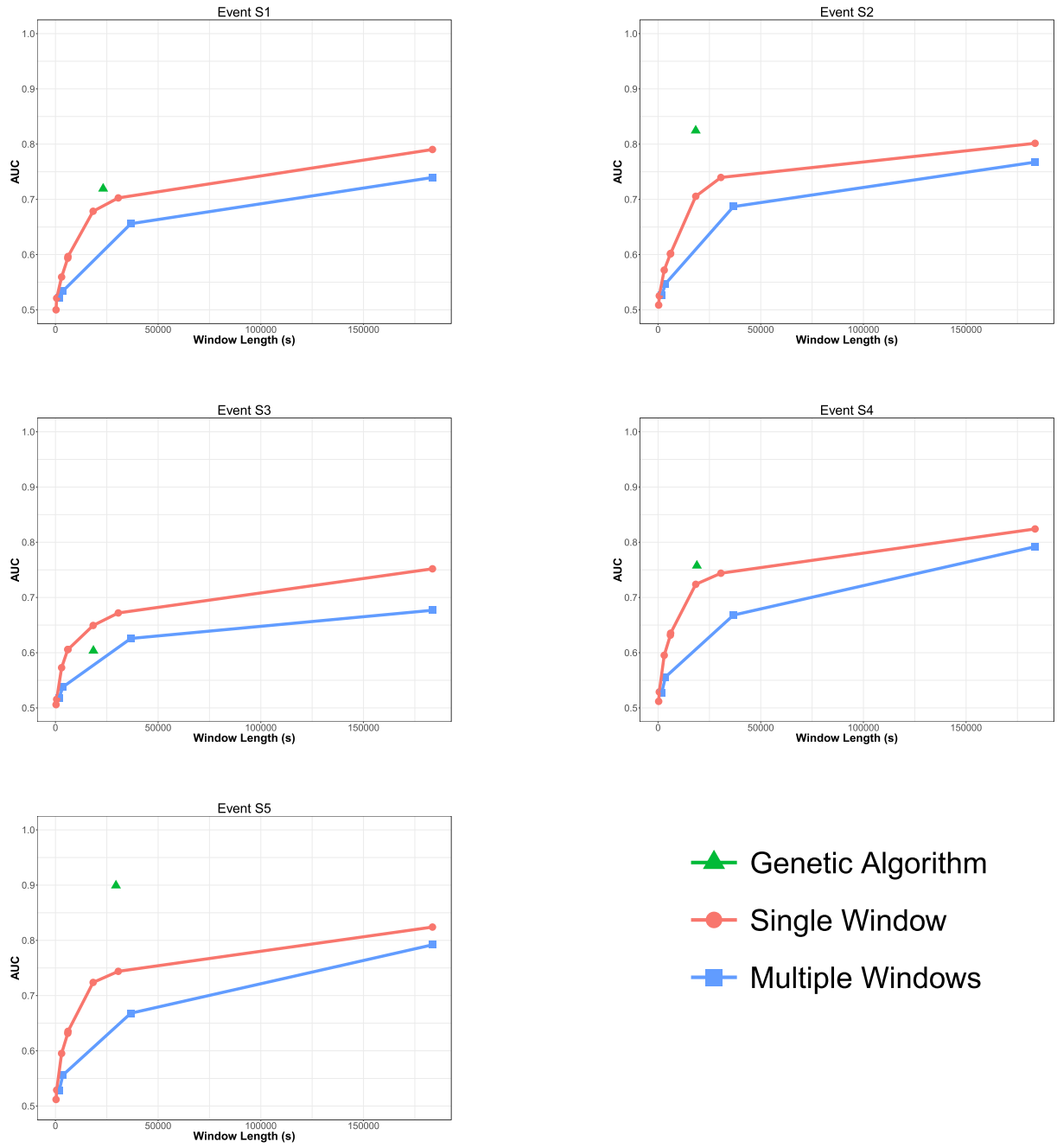


FIGURE 8. AUC vs window length results for the Genetic Algorithm versus the heuristic approach for the High Performance Computing dataset.

each event. Fig. 9 shows the AUC results for each of them. The first insight that can be extracted from the figure is how the optimal window and behavior based on it of each event is completely different (ranging from 750 seconds to more than 12000), being exactly the same training dataset. This observation suggests that, for certain datasets, a single, fixed value window for a whole dataset is not a suitable approach and it would yield subpar results. Regarding the Genetic Algorithm results, it works as intended. For events N1, N2 and N3 its solution obtains either as good a solution as larger schemes with less window length or it manages to stay between them, shrinking the window length until performance would start getting lower. But events N4 and N5 show another very interesting behavior: in both cases, the Genetic Algorithm solution obtains the best median performance with a smaller window than most of the other ones. This suggests that the algorithm has, indeed, found a complex distribution of windows that skips useless time periods and keeps the ones

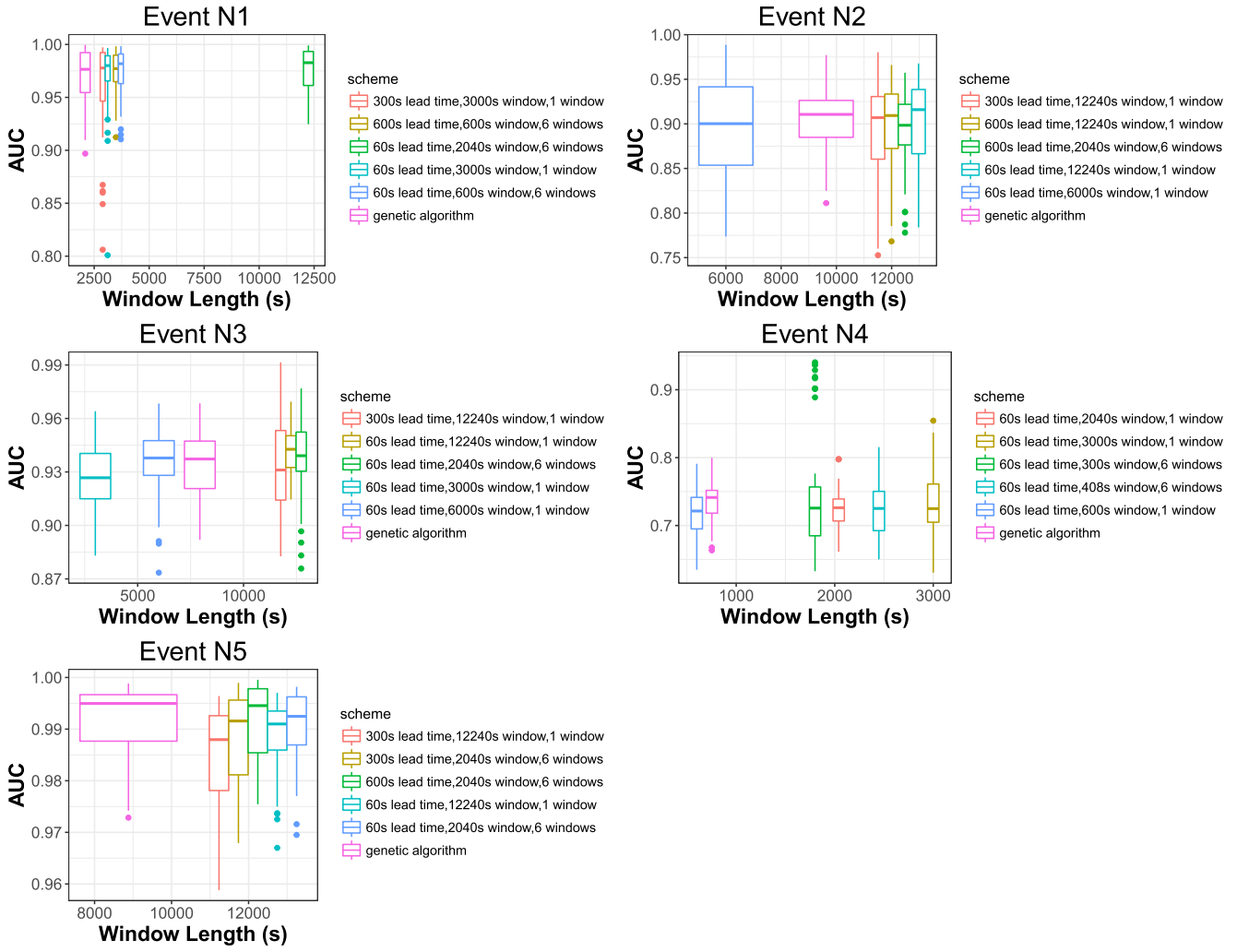


FIGURE 9. AUC vs window length results for the Genetic Algorithm versus the heuristic approach for the Networking dataset.

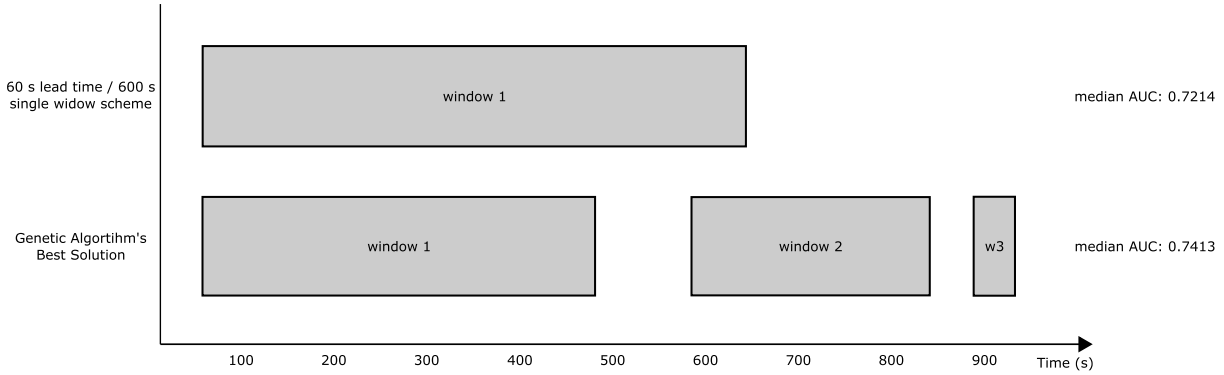


FIGURE 10. Windows selected for event N4 by the Genetic Algorithm and its closest heuristic counterpart.

that contain model-relevant information. In order to check this behavior, we show on Fig. 10 the Genetic Algorithm window for event N4 and its most similar instance in window length, a 60s of lead time and 600s single window scheme. We recommend, though, to run a similar analysis to the one we show on Fig. 7, in order to find the consensus of the algorithm and avoid the dispersion caused by random mutations. We also performed an ANOVA and a Tukey's

range test to check if the Genetic Algorithm's results were significantly better than other schemes. Utilizing the same assumptions for model comparison that we exposed on the previous subsection, the algorithm worked much better for this dataset than for the previous one: out of 36 different schemes, for the five events, our proposal was better than 25 (with a minimum of 21 for event N1 and a maximum of 30 for event N5) schemes in average (with a mean median difference of 0.14 in favor of it). For the rest of the schemes, there was no significant difference. So, even optimizing the window size, our proposal worked better (in raw performance) than the great majority of schemes tested.

There is also one area where the heuristic approach works much worse than our proposal: computational complexity. Our proposal's complexity can be defined as:

$$GP(RO(ML) + O(Fitness) + O(Selection) + O(Crossover) + O(Mutation)) \sim GPRO(ML) \quad (9)$$

where G equals the number of generations, P is the chosen population number, R is the number of cross-validation rounds and $O(ML)$ is the complexity of the chosen Machine Learning algorithm. As the number of parameters and values to test increase, the computation time of a grid search increases to unfeasible amounts: for example, for our setting and the High Performance Computing Dataset, one execution of our proposal takes around 35 minutes, whereas the grid search (using 36 different schemes) took around 7 hours and 30 minutes. Additionally, there's the fact that a simple grid search doesn't alllow for complex windows, which we saw appear on the Networking dataset.

VII. CONCLUSIONS

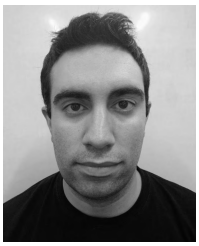
Failure Prediction, though essential for present and future networks, still requires a great deal of manual interaction to deal with new environments. Specifically, when performing a prediction process, data windowing must be carried out. There is no current solution to decide the appropriate window size and most efforts on the field only study how the window size influences their problem at hand. To solve this issue, we have proposed a novel approach, based on the usage of multiple observation windows, separated by variable time delays, automatically optimized by a customized Genetic Algorithm. This algorithm finds the solution that maximizes model performance, while trying to shrink the window size as much as possible, to save storage costs. As systems grow in complexity and the number of events to be modeled also grow, the saved storage becomes more and more significant. We have validated our solution against ten different events, extracted from two real datasets, a high performance computing dataset and a network event dataset, showing how the algorithm has consistently performed equally or outperformed a heuristic grid search, finding complex window schemes (and single windows when they were needed, which shows the algorithm is not biased in favor of multiple windows) and greatly adapting its solution to each event. While there are several caveats and areas which could conform future work, like the dependency of the variables or devising a better stopping criterion, on its current form our Genetic Algorithm solution provides a complete solution to the windowing problem. Furthermore, it can be applied to any problem in which a time window set must be decided, as nothing binds the proposal to just networks: while the windowing process and specific feature engineering of the paper focuses on Distributed Systems, the core of the approach, the usage of a Genetic Algorithm to optimize the window set, is discipline agnostic. As such, only the characteristics of the problem and not the application environment determine the appropriateness of our proposal, conferring it a great deal of flexibility and applicability.

REFERENCES

[1] F. Salfner, M. Lenk, and M. Malek, "A survey of online failure prediction methods," *ACM Comput. Surv.*, vol. 42, pp. 10:1–10:42, Mar. 2010.
[2] L. Yu, Z. Zheng, Z. Lan, and S. Coghlan, "Practical online failure prediction for blue gene/P: Period-based vs event-driven," in *Proc. IEEE/IFIP 41st Int. Conf. Dependable Syst. Netw. Workshops (DSN-W)*, Jun. 2011, pp. 259–264.
[3] H. Xu and H. Li, *The Failure Prediction of Cluster Systems Based on System Logs*. Berlin, Germany: Springer, 2013, pp. 526–537.
[4] K. B. Ferreira, A. Goudarzi, D. Arnold, D. Stefanovic, and G. Feldman, "High-end system event correlations and predictions," Nat. Nucl. Secur. Admin., Washington, DC, USA, Tech. Rep. SAND2014-4079C, 2014.
[5] E. Chuah, A. Jhumka, S. Narasimhamurthy, J. Hammond, J. C. Browne, and B. Barth, "Linking resource usage anomalies with system failures from cluster log data," in *Proc. IEEE 32nd Int. Symp. Rel. Distrib. Syst.*, Sep. 2013, pp. 111–120.
[6] H. Wang, L. Wang, Q. Yu, Z. Zheng, A. Bouguettaya, and M. R. Lyu, "Online reliability prediction via motifs-based dynamic Bayesian networks for service-oriented systems," *IEEE Trans. Softw. Eng.*, vol. 43, no. 6, pp. 556–579, Jun. 2017.
[7] X. Fu, R. Ren, S. A. McKee, J. Zhan, and N. Sun, "Digging deeper into cluster system logs for failure prediction and root cause diagnosis," in *Proc. IEEE Int. Conf. Cluster Comput. (CLUSTER)*, Sep. 2014, pp. 103–112.
[8] N. Gurumdimma, A. Jhumka, M. Liakata, E. Chuah, and J. Browne, "Towards detecting patterns in failure logs of large-scale distributed systems," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshop*, May 2015, pp. 1052–1061.
[9] C. Di Martino, *One Size Does Not Fit All: Clustering Supercomputer Failures Using a Multiple Time Window Approach*. Berlin, Germany: Springer, 2013, pp. 302–316.
[10] J. Zhong, W. Guo, and Z. Wang, "Study on network failure prediction based on alarm logs," in *Proc. 3rd MEC Int. Conf. Big Data Smart City (ICBDSC)*, Mar. 2016, pp. 1–7.
[11] T. Kimura, A. Watanabe, T. Toyono, and K. Ishibashi, "Proactive failure detection learning generation patterns of large-scale network logs," in *Proc. 11th Int. Conf. Netw. Service Manage. (CNSM)*, Nov. 2015, pp. 8–14.
[12] M. Ding and H. Tian, "Pca-based network traffic anomaly detection," *Tsinghua Sci. Technol.*, vol. 21, no. 5, pp. 500–509, Oct. 2016.
[13] I. G. Cârjeu, T. Shorrock, and M. Seeger, "Clustering IT events around common root causes," in *Proc. IEEE Int. Conf. Services Comput. (SCC)*, Jun. 2014, pp. 749–757.
[14] B. Zhu, G. Wang, X. Liu, D. Hu, S. Lin, and J. Ma, "Proactive drive failure prediction for large scale storage systems," in *Proc. IEEE Symp. Mass Storage Syst. Technol. (MSST)*, May 2013, pp. 1–5.
[15] J. Li *et al.*, "Hard drive failure prediction using classification and regression trees," in *Proc. Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Jun. 2014, pp. 383–394.
[16] L. P. Queiroz *et al.*, "A fault detection method for hard disk drives based on mixture of Gaussians and nonparametric statistics," *IEEE Trans. Ind. Informat.*, vol. 13, no. 2, pp. 542–550, Apr. 2017.
[17] I. Irrera and M. Vieira, "A practical approach for generating failure data for assessing and comparing failure prediction algorithms," in *Proc. IEEE 20th Pacific Rim Int. Symp. Dependable Comput. (PRDC)*, Nov. 2014, pp. 86–95.

[18] T. Pitakrat, D. Okanovic, A. Van Hoorn, and L. Grunske, "An architecture-aware approach to hierarchical online failure prediction," in *Proc. 12th Int. ACM SIGSOFT Conf. Qual. Softw. Archit. (QoSA)*, Apr. 2016, pp. 60–69.
[19] C. Zhou, B. Cule, and B. Goethals, "A pattern based predictor for event streams," *Expert Syst. Appl.*, vol. 42, no. 23, pp. 9294–9306, 2015.
[20] M. Gupta, J. Gao, C. C. Aggarwal, and J. Han, "Outlier detection for temporal data: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 9, pp. 2250–2267, Sep. 2014.
[21] C. Wang, H. T. Vo, and P. Ni, "An IoT application for fault diagnosis and prediction," in *Proc. IEEE Int. Conf. Data Sci. Data Intensive Syst.*, Dec. 2015, pp. 726–731.
[22] H. Li, B. Qian, D. Parikh, and A. Hampapur, "Alarm prediction in large-scale sensor networks—A case study in railroad," in *Proc. IEEE Int. Conf. Big Data*, Oct. 2013, pp. 7–14.
[23] M. Shafie-Khah, M. P. Moghaddam, and M. Sheikh-El-Eslami, "Price forecasting of day-ahead electricity markets using a hybrid forecast method," *Energy Convers. Manage.*, vol. 52, no. 5, pp. 2165–2169, 2011.
[24] H. Song, A. K. Qin, and F. D. Salim, "Multivariate electricity consumption prediction with extreme learning machine," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2016, pp. 2313–2320.
[25] A. Azadeh, S. F. Ghaderi, S. Tarverdian, and M. Saberi, "Integration of artificial neural networks and genetic algorithm to predict electrical energy consumption," *Appl. Math. Comput.*, vol. 186, no. 2, pp. 1731–1741, 2007.
[26] L. Tian and A. Noore, "On-line prediction of software reliability using an evolutionary connectionist model," *J. Syst. Softw.*, vol. 77, no. 2, pp. 173–180, 2005.
[27] C. Zhang and H. Hu, "Using PSO algorithm to evolve an optimum input subset for a SVM in time series forecasting," in *Proc. IEEE Int. Conf. Syst., Man Cybern.*, vol. 4, Oct. 2005, pp. 3793–3796.
[28] E. Hadavandi, H. Shavandi, and A. Ghanbari, "Integration of genetic fuzzy systems and artificial neural networks for stock price forecasting," *Knowl.-Based Syst.*, vol. 23, no. 8, pp. 800–808, 2010.
[29] F. Friedrichs and C. Igel, "Evolutionary tuning of multiple SVM parameters," *Neurocomputing*, vol. 64, pp. 107–117, Mar. 2005.
[30] F. H. F. Leung, H. K. Lam, S. H. Ling, and P. K. S. Tam, "Tuning of the structure and parameters of a neural network using an improved genetic algorithm," *IEEE Trans. Neural Netw.*, vol. 14, no. 1, pp. 79–88, Jan. 2003.
[31] D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE Trans. Syst., Man Cybern.*, vol. 21, no. 3, pp. 660–674, May 1991.
[32] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.
[33] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognit. Lett.*, vol. 27, no. 8, pp. 861–874, Jun. 2006.
[34] M. Kuhn and K. Johnson, *Applied Predictive Modeling*, vol. 26. New York, NY, USA: Springer, 2013.
[35] B. Schroeder and G. Gibson, "A large-scale study of failures in high-performance computing systems," *IEEE Trans. Dependable Secure Comput.*, vol. 7, no. 4, pp. 337–350, Oct. 2010.
[36] J. W. Tukey, "Comparing individual means in the analysis of variance," *Biometrics*, vol. 5, no. 2, pp. 99–114, 1949.
[37] C. E. Bonferroni, "Il calcolo delle assicurazioni su gruppi di teste," in *Studi in Onore del Professore Salvatore Ortu Carbon*, 1935, pp. 13–60.

JOSÉ MANUEL NAVARRO GONZÁLEZ received the M.Sc. degree in telecommunications engineering from the Universidad Miguel Hernndez de Elche, Spain, in 2013, and the Ph.D. degree in applied machine learning to distributed systems management from the Escuela Técnica Superior de Ingenieros de Telecomunicación, Universidad Politécnica de Madrid, Spain. His research interests are focused on managing networks and predicting their failures and the Internet of Things.



JAVIER ANDIÓN JIMÉNEZ (M'08) is currently pursuing the Ph.D. degree in proactive failure prediction with machine learning with the Escuela Técnica Superior de Ingenieros de Telecomunicación, Universidad Politécnica de Madrid (UPM). He is a Telecommunications Engineer at UPM. He has been involved in the organization of several workshops and congresses related to artificial intelligence, robotics, and programming challenges.

JUAN CARLOS DUEÑAS LÓPEZ (SM'94) received the degree in telecommunications engineering and the Ph.D. degree from the Universidad Politécnica de Madrid (UPM) in 1991and 1994, respectively. He is currently the Deputy Vice-President for research at UPM.