Optimizing Complex Event Forecasting

Vasileios Stavropoulos[1], Elias Alevizos[1], Nikos Giatrakos[2,3], Alexander Artikis[4,1]
[1]Institute of Informatics & Telecommunications, NCSR Demokritos, Greece
[2]School of Electrical & Computer Engineering, Technical University of Crete, Greece
[3]Institute for the Management of Information Systems, Athena Research Center, Greece
[4]Department of Maritime Studies, University of Piraeus, Greece
{v.stavropoulos,alevizos.elias}@iit.demokritos.gr
ngiatrakos@softnet.tuc.gr
a.artikis@unipi.gr

ABSTRACT

In Complex Event Recognition (CER), applications express business rules in the form of patterns and deploy them in a CER Engine which seeks the occurrence of such patterns on incoming streams. This is useful for practical applications which rely on the timely detection of patterns to support critical decisions. One step further, stakeholders want to act proactively, accurately forecasting the occurrence of patterns on raw streams well ahead of time to better schedule their decisions. This calls for making the transition from CER to Complex Event Forecasting (CEF). In CEF, stochastic models of future behavior are embedded into the event processing loop to project into the future the sequence of events that have occurred so far and to estimate the likelihood of the imminent occurrence of more complex patterns. CEF performance engages the stochastic model's training speed and forecast accuracy. In turn, these performance dimensions are affected by few parameters. However, CEF parameter tuning so that optimal CEF performance is achieved is a non-trivial task. This is due to the fact that there is an infinite number of possible parameter combinations, each affecting CEF performance in ways which are hard to predict. In this work, we introduce the first CEF Optimizer that gracefully automates CEF parameter tuning decisions, rapidly cherry picking good CEF configurations. We detail the novel internal architecture of our CEF Optimizer and present an elaborate empirical analysis on two applications that illustrates the effectiveness of our optimization approach.

CCS CONCEPTS

• **Information systems** → **Data management systems**; • **Theory of computation** → *Formal languages and automata theory*; • **Mathematics of computing** → **Bayesian computation**.

KEYWORDS

Complex Event Processing , Complex Event Recognition, Complex Event Forecasting, Bayesian Optimization

DEBS '22, June 27–30, 2022, Copenhagen, Denmark

ACM ISBN 978-1-4503-9308-9/22/06…$15.00
https://doi.org/10.1145/3524860.3539810

ACM Reference Format:
Vasileios Stavropoulos[1], Elias Alevizos[1], Nikos Giatrakos[2,3], Alexander Artikis[4,1]. 2022. Optimizing Complex Event Forecasting. In *The 16th ACM International Conference on Distributed and Event-based Systems (DEBS '22), June 27–30, 2022, Copenhagen, Denmark.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3524860.3539810

1 INTRODUCTION

In Complex Event Recognition (CER) expert users from an application field express business rules in the form of patterns and deploy them in a CER Engine [6, 17, 25, 27, 28]. The CER Engine is then responsible for detecting the occurrence of such patterns on tuples that stream into it, in an online fashion. This is useful for a variety of practical applications which rely on the timely detection of patterns to provide alarms, trigger subsequent business procedures or support critical decisions.

Consider, for instance, a traffic monitoring scenario on land or at sea. Moving objects continuously transmit their location to a central server. The CER Engine running at the server side is responsible for receiving simple events of position updates and combining them in order to detect higher level patterns, i.e. Complex Events (CEs), representing some business rules of interest [56, 57]. As an example, assume that the application wants to detect a CE whenever the number of times that objects move closer than D-distance to each other within an area, i.e., proximity events occur [54], exceeds a threshold T. The CER Engine will process simple position update events, detect first level CEs in case D-distance checks are true and finally output the required higher level CE when the count of first level CEs within an area exceeds T, red-flagging that area.

This flag is an alarm that triggers subsequent business procedures. On land, flags may highlight areas of a road network where drivers exhibit aggressive behavior. Therefore, subsequent business procedures would involve the placement and periodic rearrangement of traffic wardens or patrol cars. At sea, the CE may flag areas where smuggling, i.e., illegal trade of goods, is taking place and authorities can move on site to investigate the situation first hand.

Nonetheless, CER only allows for reactive measures after a pattern occurrence. In our example scenarios, the authorities can decide only after the respective CEs have occurred, which incurs a lag between the detection of the pattern and the exploitation of the respective CE detection in the field. Such a lag may impact the effect and validity of related decisions [55]. Stakeholders would prefer to act proactively, accurately forecasting the areas of smuggling

or aggressive driving behavior well ahead of time so that authorities have the time to analyze possible what-if scenarios and better schedule their reactions.

This requirement calls for making the transition from CER to CEF (Complex Event Forecasting). In CEF, a CE can be fully matched against the streaming data, in which case events are detected (as in CER), or partially matched, in which case CEs are forecast with various degrees of certainty [5]. The latter stems from stochastic models of future behavior, embedded into the event processing loop, which project into the future the sequence of events causing a partial event pattern match, to estimate the likelihood of a full match, i.e. the actual occurrence of a particular CE.

Wayeb [3–5] is one of the first CEF Engines that supports proactive event analytics. Wayeb is conceptually divided into two layers that operate in tandem. The training layer receives as input a set of patterns of interest and a training dataset of input events and uses them in order to build stochastic models predicting future events given occurred ones. Then, these models are deployed in the forecasting layer to actually forecast future events of interest.

When a CEF Engine operates in production, it forecasts events to occur within a certain, future time-horizon. Then, as time passes, it should check whether the forecast events did occur or not and use this information as feedback in a continuous training process which keeps updating the forecasting stochastic models. The behavior of the training and forecasting layers and of the engine is tuned by a handful of parameters: (1) order, i.e., how many events stochastic models should remember to base their forecasts on, (2) the desired confidence threshold to output a forecast, (3) a minimum event probability threshold so that events with lower probability are discarded and (4) a distribution smoothing parameter.

CEF performance is characterized by the training speed and the forecast accuracy for the training and the forecasting layer, respectively. Longer training periods are expected to improve the accuracy of CEF, but delay the deployment of continuously updated, accurate models in the forecasting layer [5]. To optimize CEF applications, an appropriate balance should be achieved so that good-enough trained models are deployed as soon as possible. This is a non-trivial, bi-objective optimization problem, because the number of possible parameter combinations is infinite and each combination affects performance in a non-monotonic way.

Not only the problem of optimizing CEF applications is non-trivial, but also none of the optimization approaches used for CER [1, 9, 19, 36, 41, 42, 47, 50, 53] can be adapted in the CEF context because they do not account for CEF parameter tuning, training time and optimization accuracy. Previous works on CEF [3–5] have not examined CEF parameter turing to automatically optimize performance. If we allow the end users or the applications to manually configure the CEF Engine, their decisions can be severely suboptimal, or they will not be able to decide at all, given the complexity of the problem at hand. To tackle this challenge, we introduce the first CEF Optimizer that gracefully automates CEF parameter tuning decisions. The basic concept of the proposed CEF Optimizer is to focus on learning the behavior of an objective function that incorporates the involved performance criteria. We model the performance of a CEF Engine using sample executions and Bayesian Optimization-based estimators. We, thus, remove from field experts and business analysts the burden of properly configuring a CEF Engine and we enable them to concentrate on correctly expressing business needs in the form of patterns. We present the novel internal architecture of our CEF Optimizer and an elaborate empirical analysis on two real-world applications that illustrates the ability of our optimization approach to cherry pick proper CEF configurations.

2 BACKGROUND

In this section, we present a brief overview of the framework and the engine we use, along with the basic parameters which need to be fine-tuned for the engine to work optimally. As our engine of choice, we have opted for Wayeb. Wayeb is an open-source Complex Event Recognition and Forecasting engine [3–5][1]. It is based on automata for recognition and on Markov models for forecasting. User-provided patterns are compiled into symbolic automata and these automata are subsequently given a probabilistic description by using variable-order Markov models. We also briefly present the basic concepts behind Bayesian optimization.

2.1 Complex Event Recognition

Wayeb functions by accepting as input a set of pattern definitions for the complex events that a user is interested in. The definition for each complex event must be expressed in the form of a symbolic regular expression. Symbolic regular expressions are similar to classical regular expressions, the main difference being that their terminal "symbols" are not actually symbols from a finite alphabet, but logical predicates. Thus, symbolic regular expressions, instead of checking whether a new character is equal to a terminal symbol, check whether a new "character" (in our case, characters are tuples) satisfies a given terminal predicate.

Wayeb uses the standard operators of classical regular expressions: concatenation, disjunction and Kleene-star. It can also accommodate negation and operators for different selection policies (see [25] for a discussion of selection policies). Symbolic regular expressions are defined as follows:

Definition 2.1 (Symbolic regular expression). A Wayeb symbolic regular expression (*SRE*) is recursively defined as follows:

- If ψ is a predicate, then $R := \psi$ is a symbolic regular expression, with $\mathcal{L}(\psi) = [\![\psi]\!]$, i.e., the language of ψ is the subset of all possible tuples for which ψ evaluates to TRUE;
- Disjunction / Union: If R_1 and R_2 are symbolic regular expressions, then $R := R_1 + R_2$ is also a symbolic regular expression, with $\mathcal{L}(R) = \mathcal{L}(R_1) \cup \mathcal{L}(R_2)$;
- Concatenation / Sequence: If R_1 and R_2 are symbolic regular expressions, then $R := R_1 \cdot R_2$ is also a symbolic regular expression, with $\mathcal{L}(R) = \mathcal{L}(R_1) \cdot \mathcal{L}(R_2)$, where $\cdot$ denotes concatenation. $\mathcal{L}(R)$ is then the set of all strings constructed from concatenating each element of $\mathcal{L}(R_1)$ with each element of $\mathcal{L}(R_2)$;
- Iteration / Kleene-star: If R is a symbolic regular expression, then $R' := R^*$ is a symbolic regular expression, with $\mathcal{L}(R^*) = (\mathcal{L}(R))^*$, where $\mathcal{L}^* = \bigcup_{i \geq 0} \mathcal{L}^i$ and $\mathcal{L}^i$ is the concatenation of $\mathcal{L}$ with itself i times.

[1] Wayeb source code: https://github.com/ElAlev/Wayeb.

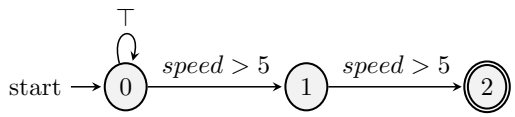


Figure 1: Streaming symbolic automaton created from the expression $R := (speed > 5) \cdot (speed > 5)$. $\top$ is a predicate which always (for every event) evaluates to TRUE. Thus, the loop on the initial state allows the automaton to skip any number of events and start recognition at any point in the stream.

- Negation / complement: If R is a symbolic regular expression, then $R' := !R$ is a symbolic regular expression, with $\mathcal{L}(R') = (\mathcal{L}(R))^c$.
- skip-till-any-match selection policy: If $R_1, R_2, \cdots, R_n$ are symbolic regular expressions, then $R' := \#(R_1, R_2, \cdots, R_n)$ is a symbolic regular expression, with $R' := R_1 \cdot \top^* \cdot R_2 \cdot \top^* \cdots \top^* \cdot R_n$, where T is a predicate that always evaluates to TRUE, regardless of the tuple on which it is applied.
- skip-till-next-match selection policy: If $R_1, R_2, \cdots, R_n$ are symbolic regular expressions, then $R' := @(R_1, R_2, \cdots, R_n)$ is a symbolic regular expression, with $R' := R_1 \cdot !(\top^* \cdot R_2 \cdot \top^*) \cdot R_2 \cdots !(\top^* \cdot R_n \cdot \top^*) \cdot R_n$.

A Wayeb expression without a selection policy implicitly follows the strict-contiguity policy, i.e., the input events involved in a match of a pattern should occur contiguously in the input stream. All the above operators, even those of selection policies, may be arbitrarily used and nested in an expression, without any limitations. This is an attractive feature of Wayeb and in contrast to other automata-based Complex Event Recognition engines which have ambiguous semantics and rules for operator usage [25].

Wayeb patterns are compiled into symbolic automata, i.e., automata whose transitions are equipped with predicates instead of symbols [18]. Every symbolic regular expression can be translated to an equivalent (i.e., with the same language) symbolic automaton [18]. As an example, consider the following pattern: $R := (speed > 5) \cdot (speed > 5)$. This simple pattern detects two consecutive events where the speed of a moving object exceeds a given threshold and could thus be used to detect speed violations (e.g., in France it is forbidden to sail with speed higher than 5 knots within 300m of the coastline). Figure 1 shows the equivalent symbolic automaton produced from this pattern.

2.2 Complex Event Forecasting

Using the automaton of Figure 1, it is possible to detect instances of speed violations. In addition to detecting pattern matches, analysts may also be interested in forecasting them. For example, an analyst may be interested in knowing whether a given moving object will violate the speed limits within the next 5 minutes. The goal of a Complex Event Forecasting engine in this case would be to evaluate whether the automaton of Figure 1 (which moves among its states as it consumes input events) is expected, with high enough confidence, to reach its final state (and thus produce a match) within the next 5 minutes.

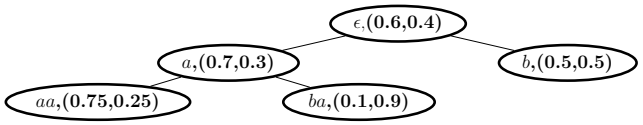


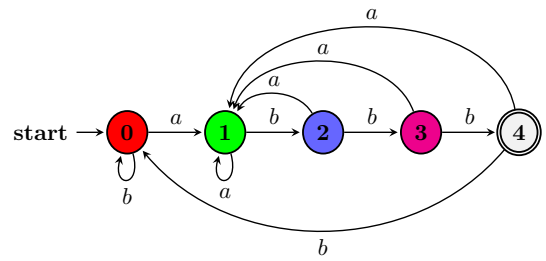
Figure 2: Example of a Prediction Suffix Tree T for $\Sigma = \{a, b\}$ and $m = 2$. Each node contains the label and the next symbol probability distribution for a and b.

Symbolic automata are sufficient to perform Complex Event Recognition. In order to perform Complex Event Forecasting, however, these automata must be given a probabilistic description. This is achieved by using variable-order Markov models (VMM). With VMMs it becomes possible to increase their order m (how many events they can remember) to higher values compared to fixed-order Markov models. It is thus possible to capture longer-term dependencies, which can lead to a better accuracy. Specifically, Prediction Suffix Trees [45, 46] are employed. Prediction Suffix Trees have been proposed in order to succinctly capture the statistical properties of sequences of symbols. Each node contains a "context" and a distribution. The distribution lets us know the probability of encountering a symbol, conditioned on the context. Figure 2 shows an example of a Prediction Suffix Tree. Note that, in our case, each "symbol" of a Prediction Suffix Tree corresponds to a predicate of the automaton for which we want to build a probabilistic model. For example, a in Figure 2 may correspond to $(speed > 5)$ of Figure 1 and b to $\neg(speed > 5)$ (negated literals are usually also included in the tree nodes, see [5] for details). Given a Prediction Suffix Tree, we can then infer how a symbolic automaton might behave in the future and when it might reach its final state and thus detect a complex event. For example, if we know that a moving object has exceeded the threshold of 5 for two consecutive events, then, according to Figure 2, the probability of this happening again is 0.75. If, additionally, we are in state 1 of Figure 1, then we know that we will detect a new complex event at the next input event with probability 0.75.

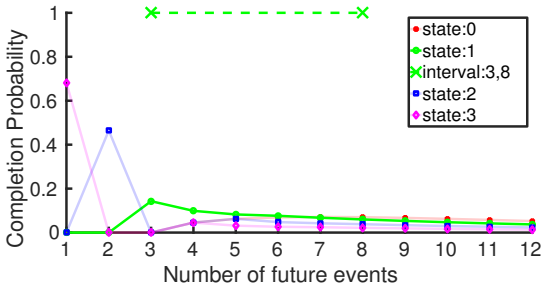
The goal is to learn a tree from a training dataset and then use it to perform online forecasting. A Prediction Suffix Tree is learned incrementally by adding new nodes only when it is necessary. The learning algorithm [46] starts with a tree having only a single node, corresponding to the empty string ϵ. Then, it decides whether to add a new context/node s by checking two conditions [46]:

- First, there must exist a symbol (predicate, in our case) σ such that $\hat{P}(\sigma \mid s) > \theta_1$ must hold, i.e., σ must appear "often enough" after the suffix s;
- Second, $\frac{\hat{P}(\sigma|s)}{\hat{P}(\sigma|suffix(s))} > \theta_2$ (or $\frac{\hat{P}(\sigma|s)}{\hat{P}(\sigma|suffix(s))} < \frac{1}{\theta_2}$) must hold, i.e., it is "meaningful enough" to expand to s because there is a significant difference in the conditional probability of σ given s with respect to the same probability given the shorter context $suffix(s)$, where $suffix(s)$ is the longest suffix of s that is different from s.

Threshold θ_1 depends on parameters α and γ, α being an approximation parameter and γ a smoothing parameter. The algorithm also discards symbols that are too rare (whose probability $P(\sigma)$ falls below a threshold $pMin$).



(a) Deterministic automaton.



(b) Waiting-time distributions and shortest interval, i.e. $[3, 8]$, exceeding a confidence threshold $\theta_{fc} = 50\%$ for state 1.

Figure 3: Automaton and waiting-time distributions for $R = a \cdot b \cdot b \cdot b$, $\Sigma = \{a, b\}$.

A Prediction Suffix Tree T can be used to calculate the so-called waiting-time distribution for every state q of an automaton A. The waiting-time distribution is the distribution of the index n, given by the waiting-time variable $W_q = inf\{n : Y_0, Y_1, ..., Y_n\}$, where $Y_0 = q$, $Y_i \in A.Q \backslash A.Q_f$ for $i \neq n$ and $Y_n \in A.Q_f$. Such a distribution lets us know the probability of reaching a final state in n transitions from any other given state. It thus allows us to estimate the probability of detecting a complex event in n transitions, since reaching a final state is equivalent to recognizing such an event. Figure 3 shows an example of an automaton and the waiting-time distributions learned from a training dataset. If the automaton is in state 2, then the probability of reaching the final state 4 for the first time in 2 transitions is $\approx 50\%$.

The waiting-time distributions can then be used to produce various kinds of forecasts. In this paper, we are interested in a type of forecasting called *CLASSIFICATION-NEXTW*. As the name suggests, the goal is to be able to answer queries of the following form: given that an automaton is in a given state, will it reach a final state within the next w transitions (or, equivalently, input events)? Such queries can be answered simply by summing the probabilities of the first w points of a distribution and if this sum exceeds a given confidence threshold θ_{fc} a "positive" forecast is emitted (meaning that a CE is indeed expected to occur); otherwise a "negative" (no CE is expected) forecast is emitted. It is important to note that positive and negative forecasts are constructed only once, as a result of the training process. When running the CEF system against a new, unknown (test) stream, the forecasts are stored in a look-up table (one for each automaton state). Whenever the automaton reaches a state, a forecast is simply retrieved from the table, without requiring any elaborate computations. The throughput of the system is thus not affected by the model complexity. On the contrary, training time can be significantly affected by the choice of the values for the parameters.

From the above discussion, it is easy to see that there are multiple hyper-parameters which need to be optimized so that a balance between forecasting accuracy and training time is achieved. For example, increasing the maximum order m of the Markov model generally leads to higher accuracy. On the other hand, this leads to deeper prediction suffix trees and more complex probabilistic models, which also increases the training time required to construct such models. The confidence threshold θ_{fc} is another crucial parameter. If θ_{fc} is very low, then the constructed model will generate many positives and thus many false positives. With high values of θ_{fc}, the result will be a large number of false negatives. We thus need to find the sweet spot for θ_{fc} that achieves the best accuracy results. It should be noted that this parameter does not have a significant effect on training time, since the computations involved are exactly the same, regardless of the specific value of θ_{fc}. The threshold *pMin* can affect both accuracy and training time. High *pMin* values mean that many "symbols" are discarded, which can lead to simpler models and decreased training times. However, excessively high *pMin* values can degrade the accuracy of our forecasts. A similar behavior can be observed with the γ parameter. These observations make it clear that the relationship between the values of the various hyper-parameters and the performance of the constructed model in terms of accuracy and training time is far from being linear. As a consequence, manual parameter optimization becomes very hard. In this paper, we employ Bayesian optimization in order to search the parameter space for the best possible hyper-parameter values.

2.3 Bayesian Optimization

A variety of scientific and industrial applications require the identification of the optimal value (maximum or minimum) of an objective function. Such optimization problems often aim at the identification of the optimal value of a black-box objective function, i.e. a function with unknown mathematical and statistical properties. Moreover, the objective function is commonly expensive to evaluate, thus limiting the number of the evaluations we are allowed to obtain. Bayesian Optimization (BO) has been used widely in the optimization of such expensive functions, as it aims at the approximation of the objective function f over the feasible set X by evaluating the function only on a limited number of sampled points.

BO optimizes an objective function f by iteratively evaluating the value of f in sampled points and constructing an estimate for the mean value of f over the set of all feasible points [11]. BO is composed of two parts: a) a statistical model used to estimate the objective function; b) an acquisition function used to efficiently sample the next points to be evaluated. A Gaussian Process Regression (GPR) approach is often followed to model the objective function. The Gaussian Process (GP) model used in the GPR approach serves as the statistical model of the optimization and provides a probability distribution that estimates the value of the objective function over the set of feasible points X [43, 51]. The model consists of a probability distribution over possible functions that fit the set

Algorithm 1 Bayesian Optimization

Place a Gaussian Process prior on f.
Evaluate f at n_o initial points. Set $n = n_0$. Update distribution based on initial evaluations.
while $n \leq N$ **do**
  Find x_n that maximizes the acquisition function over X.
  Observe $y_n = f(x_n)$.
  Update posterior distribution of f using all observed f values.
  Increment n
end while
Return the point x with the largest evaluated $f(x)$ or the point with the highest posterior mean.

of evaluated points. This distribution is updated with each new evaluation of the objective function. In particular, a GP is defined by a mean function m and a covariance function or kernel k:

$$f(x) \sim N(m(x), k(x, x'))$$

The kernel of the GP describes the smoothness of the distribution and defines the covariance between the values of the objective function between different points , i.e., how similar the evaluations of the objective function on close points are. In particular, the kernel function determines the functions that are most likely under the GP prior distribution and, thus, incorporate prior beliefs we have about the objective function. The most widely used kernel functions [23] are the Radial Basis Function (RBF) kernel, i.e.

$$k(x, x') = exp(-\frac{||x - x'||^2}{2l^2}) \quad (1)$$

and the Matérn kernel, i.e.

$$k(x, x') = \frac{1}{\Gamma(v)2^{v-1}}\left(\frac{\sqrt{2v}}{l}||x - x'||\right)^v K_v\left(\frac{\sqrt{2v}}{l}||x - x'||\right) \quad (2)$$

where l is the characteristic length scale of the kernel, K_v a modified Bessel function and Γ is the gamma function. The parameter v of the Matérn kernel controls the smoothness of the function. Small values of v identify a less smooth approximated function, while large values signal the opposite. Moreover, the kernel becomes equivalent to the RBF kernel as $v \rightarrow \infty$. The RBF kernel is used widely as it is infinitely differentiable and can, thus, approximate very smooth functions. The most common variations of the Matérn kernel are for $v = 3/2$ and $v = 5/2$, which represent once and twice differentiable functions, respectively [43].

The acquisition function allows us to select the next evaluation point in an informative manner, as it serves as a utility estimate for all feasible points. In particular, the acquisition function quantifies the contribution of the evaluation of the objective function to our estimation for each point. Commonly used acquisition functions are the Expected Improvement (EI), Probability of Improvement (PI), Lower Confidence Bound (LCB), Upper Confidence Bound (UCB) and Entropy Search (ES) [11]. Different acquisition functions estimate the importance of the evaluation of the objective function at candidate points under a different scope, balancing between the trade-off of exploration and exploitation. The acquisition functions whose primary aim is the exploration of the parameter space select points for which the estimate of the objective function is of higher uncertainty, while those that aim at the exploitation of the space sample points in which the expected mean value of the objective function is high. Hence, when exploring the parameter space, our goal is to evaluate points with high variance in uncertain regions. On the contrary, the goal of exploitation is to select points in which we are confident the objective function is high and, thus, aim at increasing the so far optimal value of the objective function.

A pseudo-code of the algorithm of BO is shown in Algorithm 1. Initially, the objective function is evaluated at n_0 randomly sampled points of the feasible set. The evaluations are used in order to obtain an initial distribution of f. Subsequently, the acquisition function is used in order to sample the next point to evaluate. In particular, the point that maximizes the acquisition function is set as the next point to evaluate. The statistical properties of the acquisition function are known and its evaluation is inexpensive. Hence, we can efficiently find the point that yields the maximum value of the acquisition function. After selecting the next point, the objective function is evaluated at the sampled point and the posterior distribution is updated based on all observed values of f. At this stage, the posterior distribution incorporates an updated understanding of the unknown objective function. During the process, the kernel of the GP allows us to estimate the value of the objective function at points that have not been evaluated yet, as it identifies all the functions under the distribution. The aforementioned process (sampling of next point → evaluation of objective function → update of posterior distribution) is repeated until we reach the declared number of total evaluated points N.

3 PROBLEM STATEMENT

We develop a CEF Optimizer with the aim of identifying the optimal configuration of Wayeb's hyperparameters for a given pattern. A candidate configuration c is defined by four parameters, as follows:

$$c = [m, \theta_{fc}, pMin, \gamma]$$

where m is the maximum order of the Prediction Suffix Tree, θ_{fc} is the desired confidence threshold of the forecast, $pMin$ is the symbol probability threshold (symbols with lower probability than $pMin$ are discarded) and γ is the distribution smoothing parameter (we have excluded α from our investigation since we have observed that its impact is typically negligible for a wide range of values). Moreover, the allowed range of each variable is set as:

$m \in [1, 5]$	$\theta_{fc} \in [0.0, 1.0]$
$pMin \in [0.0001, 0.01]$	$\gamma \in [0.0001, 0.01]$

Due to the parameter range, the feasible configurations are infinite. Moreover, it is hard to predict how a combination of parameters will affect the performance of the CEF Engine. Thus, it is important to automate the search for the optimal configuration of the engine based on the evaluation of a limited number of well performing ones. To do so, we must define a metric that provides an estimate regarding the performance of each configuration.

In order to estimate the performance of each configuration we define a scoring function which takes into consideration various metrics indicative of the performance of the framework. We define

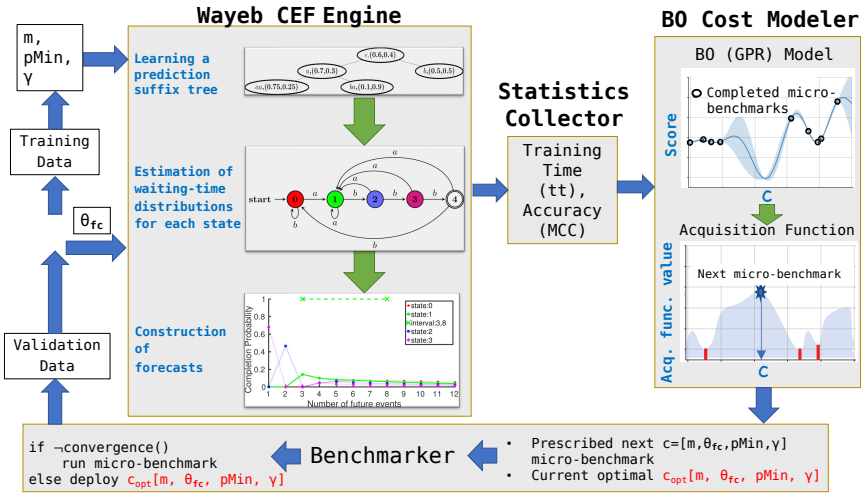


Figure 4: Overview of the CEF Optimizer.

the score of a configuration c as follows:

$$Score(c) = w_1 \times MCC(c) - w_2 \times tanh(\frac{tt(c)}{\theta_{tt}} - 1) \quad (3)$$

where MCC is the Matthews Correlation coefficient and tt the training time of the forecaster for the given configuration, while θ_{tt} is a set threshold for the training time. We use the Matthews Correlation coefficient as a measurement of the quality of the forecasts provided by Wayeb. MCC is calculated as follows:

$$MCC = \sqrt{Precision \times Recall \times Specificity \times NPV} - \sqrt{FDR \times FNR \times FPR \times FOMR} \quad (4)$$

where $NPV = \frac{TN}{TN+FN}$, $Specificity = \frac{TN}{TN+FP}$, $FDR = 1 - Precision$, $FNR = 1 - Recall$, $FPR = 1 - Specificity$ and $FOMR = 1 - NPV$.

MCC estimates the correlation of the observed and forecast pattern occurrences and its value ranges in the interval [-1,1]. A value of +1 (or −1) demonstrates the total agreement (or disagreement) between the observations and the forecasts of the framework, while a MCC equal to 0 indicates that the performance of the framework is equal to that of a random classifier. We decided to use the MCC of the forecasts in the scoring function as it returns a high value only when Wayeb succeeds in forecasting both the cases in which a pattern occurred (TP) or not (TN), thus presenting a good indicator of the performance of the engine even in imbalanced datasets. On the contrary, other popular metrics, such as the F_1 score, take into consideration only the positive instances, i.e. the occurrence of patterns in this case, and, thus, provide misleading results in imbalanced datasets, as they fail to indicate the poor performance of an engine that does not succeed in forecasting the absence of pattern occurrences [13]. Therefore, although the F_1 score is a more popular metric, we consider MCC to be more appropriate for CEF. The training time is measured in terms of milliseconds and includes the time required to learn a Prediction Suffix Tree, to estimate the waiting-time distributions for all automaton states and to construct the forecasts for *CLASSIFICATION-NEXTW*. The total training time is typically dominated by the time for estimating the waiting-time distributions, especially for high values of the order m [5].

The scoring function defined in (3) takes into consideration not only the accuracy of the forecasts, but also the amount of time required to train the model, allowing us to obtain a well-rounded estimation of the performance of each configuration in terms of both predictive accuracy and required resources. The second term of the scoring function acts as a penalty for the cost of the training time. If the training time exceeds the threshold θ_{tt} set by the user, then the total score of the configuration is reduced relatively to the cost of the training time. On the other hand, the score of the configuration is increased if the training time is lower than θ_{tt}, thus benefiting configurations that demand less training time. The scoring function can take negative values, when the cost of the training time outweighs the benefit of the accuracy of the forecast provided by the engine for the given configuration. The scoring function follows a *preference-based* approach by assigning weights to the individual objective functions (MCC and tt). Thus, the scoring function is the weighted sum of the single performance metrics and describes the performance of the framework based on the relevant preference of the end user. Hence, the scoring scheme is applicable to a wide range of applications. Based on the nature of each application a higher importance may be assigned to the accuracy of the forecaster (e.g., in applications where accurate forecasts are critical) or to the training time of the forecaster (e.g., in applications that require fast forecasts tolerating some inaccuracy).

4 CEF OPTIMIZER

Figure 4 presents our proposed CEF Optimizer. The CEF Optimizer incorporates three fundamental components along with the CEF Engine: the Benchmarker, the Statistics Collector and the BO Cost Modeler. The CEF Engine receives as input a training dataset for the training layer of Wayeb and a validation dataset for the forecasting layer. These datasets can be provided either using representative historical data of the application field or during a warm up period where the optimizer samples incoming event streams.

At the beginning of the optimization process the Benchmarker Component (bottom of Figure 4) samples, uniformly, a number n_0 of $c = [m, \theta_{fc}, pMin, \gamma]$ configurations from all possible such combinations. It then uses the sampled configurations to perform a set of initial micro-benchmarks, i.e., the CEF Engine runs. For each such micro-benchmark, the CEF Engine (in our case Wayeb) executes its training process using the training dataset. In that, a Prediction Suffix Tree is learned at the training layer of Wayeb. Then, this tree is used to calculate the waiting-time distribution for every state of an automaton at the forecasting layer of Wayeb. The accuracy of the forecasts is judged using the validation dataset (left-hand side of Figure 4).

The Statistics Collector (middle of Figure 4) is responsible for collecting statistics about the involved performance measures of accuracy (*MCC*) and training time (*tt*), respectively, from each micro-benchmark. Then, it feeds the statistics into the BO Cost Modeler Component (right-hand side of Figure 4).

The first time the BO Cost Modeler Component is used, it fits a Gaussian Process Regressor around these initial statistics. After this "fitting" phase, the BO Cost Modeler uses the acquisition function to prescribe the next micro-benchmark, i.e., a new $c = [m, \theta_{fc}, pMin, \gamma]$ configuration, as input to the Benchmarker. New configurations are then sampled and evaluated in an iterative manner, until we reach a satisfactory configuration. What makes BO effective is that this exploration is not exhaustive, ignoring and undersampling regions of the parameter space from which previous samples did not produce promising results.

Each time the Benchmarker receives a new request for executing a micro-benchmark, it queries back the BO Cost Modeler to derive the current optimal (max) score (Equation 3) along with the optimal CEF Engine configuration, c_{opt} in Figure 4, which yields that score. This is illustrated at the bottom right part of Figure 4 with the bi-directional arrow between the Benchmarker and the BO Cost Modeler. The bottom left part of Figure 4 shows the test used by the Benchmarker to decide how the optimization process should continue. If the optimization process has converged, i.e., the optimal score has not changed significantly during recent micro-benchmarks, the Benchmarker does not execute another micro-benchmark. Instead, it exploits the current optimal configuration, deploying the CEF Engine with c_{opt} parameters and the tuned CEF Engine is put in production. Otherwise, the Benchmarker configures the CEF Engine to execute another micro-benchmark, the one just prescribed by the BO Cost Modeler. In the latter case, the optimization process follows again the steps of single micro-benchmark execution → statistics collection → GPR update, <prescribed micro-benchmark, c_{opt}> pair extraction → convergence test.

5 EMPIRICAL ANALYSIS

5.1 Experimental Setup

We evaluated the performance of the CEF Optimizer on the domains of maritime monitoring and credit card fraud management.

The score of each Wayeb configuration is estimated based on a 5-fold cross validation. The training time is calculated as the required time for the construction of the Prediction Suffix Tree, the Waiting-Time Distribution and for the production of the forecasts. In each repetition of the cross-fold validation, the Statistics Collector of the optimizer measures the number of True Positive (*TP*), True Negative (*TN*), False Positive (*FP*), False Negative (*FN*) forecasts. The accuracy of the forecasts of each configuration is evaluated as a macro-average of the performance of each cross-validation, i.e. we calculate the *MCC* as described in Equation (4) using the cumulative sum of the *TP*, *TN*, *FP*, *FN*.

We simulated the randomness in the various steps of the empirical analysis, i.e. the random selection of the set n_0 of initial points (micro-benchmarks), using a pseudorandom number generator. The use of different random number generator seeds in our experiments allowed us not only to examine the application of the optimizer under different experimental conditions but also to compare the performance of different approaches under the same conditions. The selection of the initial micro-benchmarks was based on uniform random sampling. Moreover, the BO Cost Modeler applies a Gaussian Process Regressor (GPR) with a Matérn ($\nu = 3/2$) kernel (see Equation (2)) and the Expected Improvement (EI) acquisition function in order to estimate the objective function. The EI acquisition function considers both the probability and the magnitude of the potential improvement yielded by a candidate configuration.

With respect to the scoring function, we assigned equal importance to the accuracy of the forecasts and the training time. Hence, we set $w_1 = w_2 = 0.5$ in Equation (3).

5.2 Maritime Monitoring

In the first setting, we tested our CEF Optimizer against a real, open dataset from the field of maritime monitoring[2]. During their course, vessels emit AIS (Automatic Identification System) messages that relay navigational information about the vessels' course, such as position, speed and heading. These messages are collected by AIS base stations along the coastline, and are processed with the purpose of monitoring the vessels and detecting interesting patterns in their behavior. The dataset used in our experiments contains AIS messages transmitted by vessels sailing around the port of Brest, France, from 1 October 2015 to 31 March 2016 [44].

We evaluated the CEF Optimizer on a pattern defining when a vessel approaches the main port of Brest. The Wayeb expression is:

$$R := (\neg InsidePort(Brest))^* \cdot (\neg InsidePort(Brest)) \cdot (\neg InsidePort(Brest)) \cdot (InsidePort(Brest))$$

When a vessel has a distance of less than 5 km from the port of Brest, $InsidePort(Brest)$ evaluates to TRUE. In the pattern, an entrance to the port is defined as a sequence of at least 3 consecutive events. At least two events check whether the vessel is outside the port and only the last event must satisfy the $InsidePort(Brest)$ predicate. We require that there are at least 2 events where the vessel was outside the port, in order to avoid detecting exits, long stays inside the port and "noisy" entrances.

In the first set of experiments, we compared the performance of the CEF Optimizer against a sweep search in the parameter space. Sweep search consisted of the evaluation of a finite set of configurations. Wayeb's hyperparameters span continuous value ranges. Thus, it is impossible to exhaustively evaluate all the feasible configurations of the parameter space and find the exact optimal configuration yielding the highest score. Our sweep search

[2] https://zenodo.org/record/1167595#.YkFm2S8Rqgc

Variable	Lower Limit	Upper Limit	Step
m	1	5	1
θ_{fc}	0.0	1.0	0.1
$pMin$	0.0001	0.01	0.002475
γ	0.0001	0.01	0.002475

Table 1: Sampling of the discrete grid of sweep search

approach discretizes Wayeb's parameter space and evaluates only a finite set of feasible configurations. In particular, the sweep search benchmark consists of the evaluation of 1, 375 configurations. These configurations are the points of the grid of the discrete version of the parameter space, based on the sampling presented in Table 1.

Figure 5a illustrates the scores of the evaluated configurations of the sweep search. In particular, Figure 5a presents the sampled areas of the 4-dimensional configuration parameter space as a grid of $2D$ plots. The axes along each $2D$ subplot represent the confidence threshold θ_{fc} and $pMin$ (symbols that appear with probability below $pMin$ are discarded) of the configurations. The subplots of each row of the grid are defined by a common maximum order m of the Prediction Suffix Tree and each column of the grid is defined by a different sampled value for the smoothing parameter γ of the configurations. As presented in Figure 5a, we observe an improvement in the score of the configurations as we increase the value of the smoothing parameter γ of the configurations. Moreover, we observe that the configurations with low to medium confidence thresholds θ_{fc} and with orders m ranging from 1 to 3 perform noticeably better than the rest of the configurations of the grid of sweep search.

Our optimizer identifies the configuration of Wayeb that yields the highest score, i .e. the optimal configuration of the CEF Engine. The optimization is based on the construction of an estimation of the score yielded by the feasible configurations of the parameter space. The updated probability distribution obtained at the end of each iteration of the optimization provides an expected mean score and standard deviation for each feasible configuration. The probability distribution is used in order to select in an informative manner the next configuration to evaluate based on the acquisition function. Moreover, the distribution acts as an indicator of the highest estimated score that can be achieved, as stated in the end of Algorithm 1. Figure 5b represents the expected mean score of the CEF Optimizer as a grid of $2D$ plots.

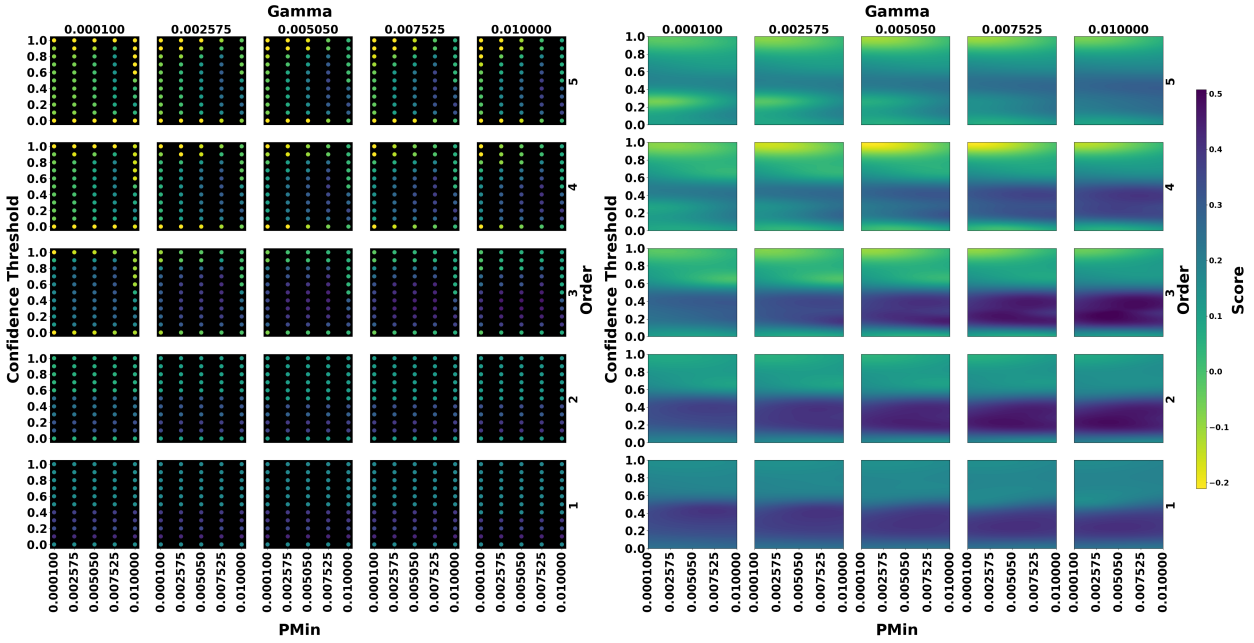
We observe that the areas estimated as more promising by our optimizer, i.e. the areas containing configurations expected to have the highest scores, are comparable to the areas found to contain the best performing configurations during sweep search. This is a significant result, since, in these experiments, the CEF Optimizer implemented 32 micro-benchmarks, while sweep search implemented 1, 375 micro-benchmarks. In other words, our CEF Optimizer computes the optimal configurations with minimal resource usage. Both the optimizer and sweep search found that configurations with order $m = 3$, θ_{fc} ranging from 0.1 to 0.5 and higher γ values yielded the highest scores. Moreover, we observe that the configurations of higher orders m yield higher scores for $pMin$ ranging from 0.4 to 0.8. These optimal configurations succeed in balancing the finer accuracy offered by a higher order model with the increased training times required for the construction of such high-order models. In addition, the optimizer identifies not only the most effective configurations but also the worst-performing ones, as indicated by the lighter shaded areas of Figure 5b.

Figure 6a presents the highest scores yielded by the configurations computed by: (a) sweep search and (b) the CEF Optimizer for five different random seeds of initial micro-benchmarks. As mentioned earlier, the optimizer discovers configurations that yield scores comparable to the score obtained by the optimal configuration of the discrete sweep search. Sampling in the continuous space, as opposed to searching in a discretized version of the search space, allows the CEF Optimizer to outperform, in some cases, sweep search. Moreover, recall that the CEF Optimizer achieves these scores by performing only 32 micro-benchmarks in comparison to the 1, 375 micro-benchmarks of sweep search.

In our second set of experiments, we compared the performance of our CEF Optimizer against a scenario where configurations are sampled uniformly. Figure 6b illustrates the improvement of the CEF Optimizer over the highest score achieved when selecting configurations randomly. The optimizer is compared to the random sampling of configurations in each of the five random seeds used.

We observe that the optimizer discovers more effective configurations, i.e. discovers a configuration that yields a higher score, in all examined random seeds. Such a performance gain leads to more accurate and timely forecasts, paving the way for proactive decision-making. The optimizer exploits the information provided by the acquisition function of the BO Cost Modeler and selects the next micro-benchmark to conduct in an informative manner, thus achieving a better guided search and ultimately discovering higher-performing configurations.

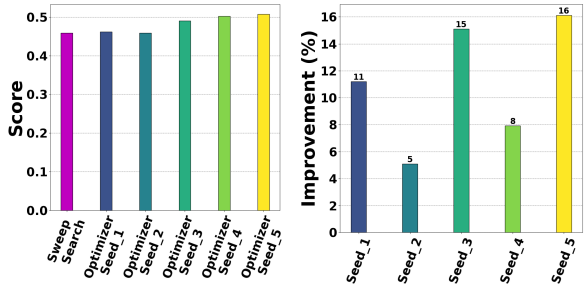
According to Algorithm 1, a set of initial micro-benchmarks/ points n_0 is used in order to place an initial probability distribution over the objective function. These initial micro-benchmarks of the optimizer are not sampled based on the acquisition function, but are chosen at random. In our third set of experiments, we examined the performance of the optimizer using a varying number of initial micro-benchmarks. Figure 7a illustrates the estimated highest score obtained at the end of the optimization process. The first bar of each seed illustrates the highest estimated score after the initialization of the CEF Optimizer using 2 micro-benchmarks, while the second, third and fourth bars of each seed use 4, 8 and 16 initial micro-benchmarks, respectively. We observe that the versions of the CEF Optimizer which required 8 and 16 initial micro-benchmarks achieve the highest score across all seeds in comparison to the versions requiring 2 and 4 initial micro-benchmarks. The evaluation of more initial micro-benchmarks allows the optimizer to obtain a clearer understanding of the configuration parameter space, allowing it to focus on areas with high scoring configurations using the acquisition function. The version of the CEF Optimizer that is initialized with 8 micro-benchmarks presents the best performance in the majority of the random seeds examined. It is, thus, important to choose wisely the allocation of the total available micro-benchmarks between the initial micro-benchmarks that are chosen at random (used for the construction of an initial distribution) and the micro-benchmarks sampled based on the BO Cost Modeler, i.e. the configurations selected exploiting the so-far available information by means of the acquisition function.



(a) The 1,375 scores of sweep search. To aid legibility, we employed a black background.

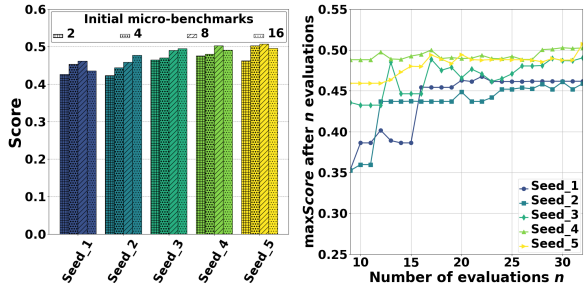
(b) Estimated performance of the different areas of the configuration parameter space by the CEF Optimizer.

Figure 5: Maritime monitoring: Comparison of sweep search and the CEF Optimizer. Darker colors represent configurations that yield higher scores. The colorbar applies to the discrete points of figure (a) and the surface of figure (b).



(a) **(b)**

Figure 6: Maritime monitoring: (a) The best scores obtained by sweep search and the CEF Optimizer. (b) Improvement of the score by the CEF Optimizer over the best score obtained by random evaluations.



(a) **(b)**

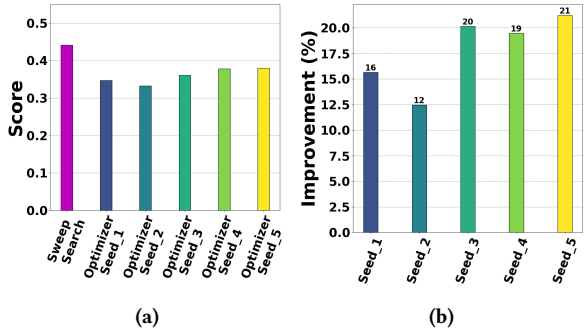
Figure 7: Maritime monitoring: (a) Highest estimated score discovered by the CEF Optimizer initialized with a varying number of micro-benchmarks. (b) Convergence of the optimization process.

Figure 7b displays the highest score achieved by the optimizer during the optimization process. In these experiments, the initial number of micro-benchmarks was set to 8, while the optimizer used the acquisition function for 24 more micro-benchmarks. Figure 7b shows that the optimizer converges even before the 32nd micro-benchmark. The optimizer, thus, fine-tunes the CEF Engine early in the optimization process, leading to significant gains in resource usage, as opposed to, say, an exhaustive search, such as the sweep search benchmark presented earlier.

5.3 Credit Card Fraud Management

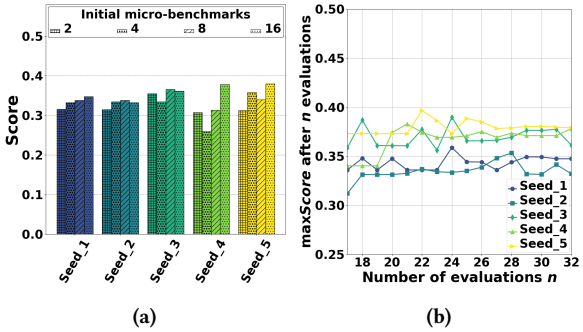
In the second setting of our experiments we used a synthetic dataset from the domain of credit card fraud management provided by Feedzai [3]. Each event of the dataset represents a credit card transaction and is accompanied by details of the transaction, such as the credit card ID, the amount and time of the transaction, etc. We evaluated our CEF Optimizer in a pattern representing a fraudulent

[3] https://feedzai.com/



(a) (b)

Figure 8: Credit card fraud management: (a) The best scores obtained by sweep search and the CEF Optimizer. (b) Improvement of the score by the CEF Optimizer over the best score obtained by random evaluations.



(a) (b)

Figure 9: Credit card fraud management: (a) Highest estimated score discovered by the CEF Optimizer initialized with a varying number of micro-benchmarks. (b) Convergence of the optimization process.

behavior as a sequence of consecutive increasing transactions [8]. The Wayeb expression is:

$$R := (amountDiff > 0) \cdot (amountDiff > 0) \cdot (amountDiff > 0) \cdot (amountDiff > 0) \cdot (amountDiff > 0) \cdot (amountDiff > 0) \cdot (amountDiff > 0) \cdot (amountDiff > 0)$$

Each event is enriched with an additional attribute *amountDiff* (on top of the standard attributes of credit card ID, etc.) that is equal to the difference between the amount spent by the current and immediately previous transactions. When the amount spent on the current transaction is greater than the amount spent on the immediately previous transaction, $(amountDiff > 0)$ evaluates to TRUE. The examined pattern defining the credit card fraud requires the occurrence of 8 consecutive increasing transactions.

In our experiments, the CEF Optimizer succeeds in discovering configurations that yield scores comparable to that of the optimal discovered by sweep search, as seen in Figure 8a. Similar to the maritime use case, the optimizer discovers high-performing configurations requiring only 32 micro-benchmarks, considerable fewer than the 1,375 micro-benchmarks required by sweep search. Thus, the optimizer succeeds in approximating the highest scores achieved by a sweeping exploration of the configuration parameter space while requiring significantly fewer micro-benchmarks.

In the second set of our experiments on credit card fraud management, we compare the performance of the CEF Optimizer against the evaluation of randomly uniformly sampled configurations. Figure 8b presents the improvement of the CEF Optimizer over the highest score achieved by the random sampling of configurations. We observe that the CEF Optimizer discovers better-performing configurations when compared to the evaluation of randomly selected configurations in all examined random seeds.

Figure 9a demonstrates the highest estimated scores obtained by different versions of the CEF Optimizer at the end of the optimization process. The different versions of the optimizer use a varying number of initial micro-benchmarks. We observe that the highest estimated score in the majority of the examined seeds is achieved by the version of the optimizer that requires 16 initial micro-benchmarks.

Figure 9b presents the convergence of the highest estimated score of the CEF Optimizer throughout the optimization process. In these experiments, the optimizer used 16 initial micro-benchmarks for the creation of an initial probability distribution and proceeded with another 16 micro-benchmarks using the acquisition function. Figure 9b shows that 32 micro-benchmarks are sufficient to reach convergence.

6 RELATED WORK

In the CER domain, most optimization approaches aim at increasing the CER Engine's throughput, i.e., number of tuples being processed per time unit [1, 9, 19, 36, 41, 42, 47, 50, 53]. Secondary optimization metrics relate to reducing processing latency or to optimally control memory utilization. The concepts used in these techniques essentially adapt common query optimization practices, such as early evaluation of predicates and query rewriting, to the CER context. Flouris et al [21] provide a concise summary on such techniques. Giatrakos et al [25] discuss techniques for efficiently executing CER over parallel and geo-distributed settings. None of these approaches deals with Complex Event Forecasting.

Forecasting has been a very active research direction in various fields. Time-series forecasting is a well-known example [37]. Time-series forecasting typically focuses on streams of (mostly) real-valued variables and the goal is to forecast relatively simple patterns. Sequence prediction is another field with a long history of contributions [10, 16, 45, 46, 52]. In this case, the goal is to predict the next symbol(s) to occur in sequences constructed from finite alphabets. However, pattern forecasting is outside the scope of these efforts. A similar line of work has developed recently concerning event sequence prediction and point-of-interest recommendations through the use of neural networks [12, 33]. The focus is again on input (and not complex) event forecasting. A significant number of forecasting methods comes from the field of temporal pattern mining, where patterns are usually defined either as association rules [2] or as frequent episodes [34]. Typical examples may be found in [14, 31, 49, 59]. The proposed methods from this field try to forecast simple patterns, defined either as strict sequences or as sets of input events. They also typically make the assumption that input streams are composed of symbols from a finite alphabet. The field of process

mining (and process prediction) is more closely related to CEF [48]. Processes are typically defined as transition systems (e.g., automata or Petri nets) and are used to monitor a system, e.g., for conformance testing. Process mining attempts to automatically learn a process from a set of traces, i.e., a set of activity logs. Since 2010, a significant body of work has appeared, targeting process prediction, where the goal is to forecast if and when a process is expected to be completed (for surveys, see [22, 35]). An important difference is that processes are usually given directly as transition systems, whereas CER patterns are defined in a declarative manner. The transition systems defining processes are usually composed of long sequences of events. On the other hand, CER patterns are shorter, may involve Kleene-star, iteration operators (usually not present in processes) and may even be instantaneous. Another important difference is that process prediction focuses on traces, which are complete, full matches, whereas CEF focuses on continuously evolving streams which may contain many irrelevant events. The limitation that is common to all of the above proposals is that they do not target complex events defined in a declarative manner through a proper language. They focus either on input events or on very simple patterns. Additionally, they often cannot accommodate multiple variables of different types (both numerical and categorical).

Complex Event Forecasting (CEF) attempts to address these challenges, as described in various conceptual proposals [15, 20, 24]. In [39], Hidden Markov Models (HMM) are used to construct a probabilistic model for the behavior of a transition system describing a complex event. Automata (for describing complex events) and Markov chains (for building a probabilistic model) are used in [38], the first concrete attempt at CEF. Wayeb, the CEF Engine that we use, follows a similar approach, but it uses high-order Markov models [3–5]. Recently, a different approach was proposed in [32]. Knowledge graphs are used to encode events and their timing relationships. However, this method would more properly fall under the category of input event forecasting and it does not support a language with which to define complex events. None of the above mentioned proposals attempts to automatically optimize their hyper-parameters, without resorting to an exhaustive search. We present the first such attempt.

The work most relevant to ours is that of EasyFlinkCEP [26] which employs Bayesian Optimization (BO) to auto-tune the optimal parallelism of FlinkCEP programs based on the specified pattern, selection and consumption policies as well as window specifications [25, 26]. EasyFlinkCEP does not support forecasting and does not consider predictive accuracy. Instead, it focuses on optimizing system-oriented metrics, such as throughput, making proper use of available resources. Therefore, it lacks the ability to provide the means for proactive measures to the involved applications. On the contrary, our optimizer tunes CEF parameters to achieve balance between training speed and forecasting accuracy.

Various approaches have been proposed in the literature for optimising Big Data management systems. CherryPick [7] aims at creating accurate performance models for finding (near-)optimal deployments (e.g., number and types of VM instances) that satisfy a performance target. It is low-overhead and applies BO to a few samples of deployment configurations in order to obtain a performance prediction for the full set of potential setups. Kunjir and Babu [30] compare black box methods, including BO and Deep Distributed Policy Gradient, against a proposed white-box algorithm to determine close-to-optimal tuning for memory-based analytics.

Broader machine learning techniques have been used for fine-tuning efficient cloud configurations and for workload estimation or query performance prediction of data management systems in the cloud. Seagull [40] uses machine learning models to predict customer load per server, and optimize service operations for database systems in the cloud. CBTune [58] uses Reinforcement Learning and utilizes a deep deterministic policy gradient method to find the optimal database instance configurations in high-dimensional continuous spaces. CDBTune adopts a try-and-error strategy to learn knob settings with a limited number of samples for initial training. It also adopts a reward-feedback mechanism in Reinforcement Learning to accelerate the convergence of the model. Herodotou et al [29] provide a concise survey of machine learning-based techniques, including BO, for tuning Big Data management systems.

7 SUMMARY & FUTURE WORK

We introduced the first CEF Optimizer tailored to automatically tune Complex Event Forecasting engines to be effectively deployed in real-world application scenarios. In that, we achieved to remove from field experts and business analysts the burden of properly configuring the CEF Engine and we allowed them to focus on correctly expressing business needs in the form of patterns. Our CEF Optimizer achieved appropriate balance between the conflicting optimization objectives of reducing training time and increasing forecasting accuracy by learning the behavior of an objective function that incorporates the involved performance criteria. We proposed a novel architecture for the CEF Optimizer and we reasoned about our design choices. We applied the CEF Optimizer on a state-of-the-art CEF Engine, namely Wayeb. A detailed empirical analysis on two real-world scenarios from the maritime and financial domain demonstrated the applicability and the effectiveness of the CEF Optimizer in prescribing near-optimal CEF Engine configurations using only a sample of CEF Engine runs, out of an infinite set.

Our future work concentrates on extending our CEF Optimizer to render it capable not only of taking optimization decisions before deploying the CEF Engine, but also to perform adaptive parameter tuning of the CEF Engine at runtime. This is useful in order to handle scenarios of highly volatile event distributions and statistical properties of raw streams.

ACKNOWLEDGMENTS

This work has received funding from the EU Horizon 2020 programs INFORE under grant agreement No 825070 and VesselAI under grant agreement No 957237.

REFERENCES

[1] Jagrati Agrawal, Yanlei Diao, Daniel Gyllstrom, and Neil Immerman. 2008. Efficient Pattern Matching over Event Streams. In *SIGMOD*.
[2] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. 1993. Mining Association Rules between Sets of Items in Large Databases. In *SIGMOD*.
[3] Elias Alevizos, Alexander Artikis, and George Paliouras. 2017. Event Forecasting with Pattern Markov Chains. In *DEBS*.
[4] Elias Alevizos, Alexander Artikis, and George Paliouras. 2018. Wayeb: a Tool for Complex Event Forecasting. In *LPAR*.
[5] Elias Alevizos, Alexander Artikis, and Georgios Paliouras. 2022. Complex event forecasting with prediction suffix trees. *VLDB J.* 31, 1 (2022), 157–180.

[6] Elias Alevizos, Anastasios Skarlatidis, Alexander Artikis, and Georgios Paliouras. 2017. Probabilistic Complex Event Recognition: A Survey. *ACM Comput. Surv.* 50, 5 (2017), 71:1–71:31.

[7] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. 2017. CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics. In *NSDI*.

[8] Alexander Artikis, Nikos Katzouris, Ivo Correia, Chris Baber, Natan Morar, Inna Skarbovsky, Fabiana Fournier, and Georgios Paliouras. 2017. A Prototype for Credit Card Fraud Management: Industry Paper. In *Proceedings of DEBS*. ACM.

[9] Roger S. Barga, Jonathan Goldstein, Mohamed Ali, and Minsheng Hong. 2007. Consistent Streaming Through Time: A Vision for Event Stream Processing. In *CIDR*.

[10] Ron Begleiter, Ran El-Yaniv, and Golan Yona. 2004. On Prediction Using Variable Order Markov Models. *J. Artif. Intell. Res.* 22 (2004), 385–421.

[11] Eric Brochu, Vlad M. Cora, and Nando de Freitas. 2010. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning.

[12] Buru Chang, Yonggyu Park, Donghyeon Park, Seongsoon Kim, and Jaewoo Kang. 2018. Content-Aware Hierarchical Point-of-Interest Embedding Model for Successive POI Recommendation. In *IJCAI*.

[13] Davide Chicco and Giuseppe Jurman. 2020. The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC Genomics* 21, 1 (jan 2020).

[14] Chung-Wen Cho, Yi-Hung Wu, Show-Jane Yen, Ying Zheng, and Arbee L. P. Chen. 2011. On-line rule matching for event prediction. *VLDB J.* 20, 3 (2011), 303–334.

[15] Maximilian Christ, Julian Krumeich, and Andreas W. Kempa-Liehr. 2016. Integrating Predictive Analytics into Complex Event Processing by Using Conditional Density Estimations. In *EDOC Workshops*.

[16] John G. Cleary and Ian H. Witten. 1984. Data Compression Using Adaptive Coding and Partial String Matching. *IEEE Trans. Communications* 32, 4 (1984), 396–402.

[17] Gianpaolo Cugola and Alessandro Margara. 2012. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.* 44, 3 (2012), 15:1–15:62.

[18] L D'Antoni and M Veanes. 2017. The Power of Symbolic Automata and Transducers. In *CAV (1)*.

[19] Alan Demeers, Johannes Gehrke, Biswanath Panda, Mirek Riedewald, Varun Sharma, and Walker White. 2007. Cayuga: A General Purpose Event Monitoring System. In *CIDR*.

[20] Yagil Engel and Opher Etzion. 2011. Towards proactive event-driven computing. In *DEBS*.

[21] Ioannis Flouris, Nikos Giatrakos, Antonios Deligiannakis, Minos N. Garofalakis, Michael Kamp, and Michael Mock. 2017. Issues in complex event processing: Status and prospects in the Big Data era. *J. Syst. Softw.* 127 (2017), 217–236.

[22] Chiara Di Francescomarino, Chiara Ghidini, Fabrizio Maria Maggi, and Fredrik Milani. 2018. Predictive Process Monitoring Methods: Which One Suits Me Best?. In *BPM*.

[23] Peter I. Frazier. 2018. A Tutorial on Bayesian Optimization.

[24] Lajos Jeno Fülöp, Árpád Beszédes, Gabriella Toth, Hunor Demeter, László Vidács, and Lóránt Farkas. 2012. Predictive complex event processing: a conceptual framework for combining complex event processing and predictive analytics. In *BCI*.

[25] Nikos Giatrakos, Elias Alevizos, Alexander Artikis, Antonios Deligiannakis, and Minos N. Garofalakis. 2020. Complex event recognition in the Big Data era: a survey. *VLDB J.* 29, 1 (2020), 313–352.

[26] Nikos Giatrakos, Eleni Kougioumtzi, Antonios Kontaxakis, Antonios Deligiannakis, and Yannis Kotidis. 2021. EasyFlinkCEP: Big Event Data Analytics for Everyone. In *CIKM*.

[27] Alejandro Grez, Cristian Riveros, Martín Ugarte, and Stijn Vansummeren. 2020. On the Expressiveness of Languages for Complex Event Recognition. In *ICDT*.

[28] Alejandro Grez, Cristian Riveros, Martín Ugarte, and Stijn Vansummeren. 2021. A Formal Framework for Complex Event Recognition. *ACM Trans. Database Syst.* 46, 4 (2021), 16:1–16:49.

[29] Herodotos Herodotou, Yuxing Chen, and Jiaheng Lu. 2020. A Survey on Automatic Parameter Tuning for Big Data Processing Systems. *ACM Comput. Surv.* 53, 2 (2020), 43:1–43:37.

[30] Mayuresh Kunjir and Shivnath Babu. 2020. Black or White? How to Develop an AutoTuner for Memory-based Analytics. In *SIGMOD*.

[31] Srivatsan Laxman, Vikram Tankasali, and Ryen W. White. 2008. Stream prediction using a generative model based on frequent episodes in event sequences. In *KDD*.

[32] Yan Li, Tingjian Ge, and Cindy X. Chen. 2020. Data Stream Event Prediction Based on Timing Knowledge and State Transitions. *Proc. VLDB Endow.* 13, 10 (2020), 1779–1792.

[33] Zhongyang Li, Xiao Ding, and Ting Liu. 2018. Constructing Narrative Event Evolutionary Graph for Script Event Prediction. In *IJCAI*.

[34] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. 1997. Discovery of Frequent Episodes in Event Sequences. *Data Min. Knowl. Discov.* 1, 3 (1997), 259–289.

[35] Alfonso Eduardo Márquez-Chamorro, Manuel Resinas, and Antonio Ruiz-Cortés. 2018. Predictive Monitoring of Business Processes: A Survey. *IEEE Trans. Services Computing* 11, 6 (2018), 962–977.

[36] Yuan Mei and Samuel Madden. 2009. Zstream: A Cost-based Query Processor for Adaptively Detecting Composite Events. In *SIGMOD*.

[37] Douglas C Montgomery, Cheryl L Jennings, and Murat Kulahci. 2015. *Introduction to time series analysis and forecasting*. John Wiley & Sons.

[38] Vinod Muthusamy, Haifeng Liu, and Hans-Arno Jacobsen. 2010. Predictive publish/subscribe matching. In *DEBS*.

[39] Suraj Pandey, Surya Nepal, and Shiping Chen. 2011. A test-bed for the evaluation of business process prediction techniques. In *CollaborateCom*.

[40] Olga Poppe, Tayo Amuneke, and et al. 2020. Seagull: An Infrastructure for Load Prediction and Optimized Resource Allocation. *Proc. VLDB Endow.* 14, 2 (2020), 154–162.

[41] Olga Poppe, Chuan Lei, Elke A. Rundensteiner, and Dan Dougherty. 2016. Context-aware Event Stream Analytics. In *EDBT*.

[42] Yingmei Qi, Lei Cao, Medhabi Ray, and Elke A. Rundensteiner. 2014. Complex Event Analytics: Online Aggregation of Stream Sequence Patterns. In *SIGMOD*.

[43] Carl Edward Rasmussen and Christopher K I Williams. 2005. *Gaussian processes for machine learning*. MIT Press.

[44] Cyril RAY, Richard DRÉO, Elena CAMOSSI, and Anne-Laure JOUSSELME. 2018. *Heterogeneous Integrated Dataset for Maritime Intelligence, Surveillance, and Reconnaissance*.

[45] Dana Ron, Yoram Singer, and Naftali Tishby. 1993. The Power of Amnesia. In *NIPS*.

[46] Dana Ron, Yoram Singer, and Naftali Tishby. 1996. The Power of Amnesia: Learning Probabilistic Automata with Variable Memory Length. *Machine Learning* 25, 2-3 (1996), 117–149.

[47] Nicholas Poul Schultz-Møller, Matteo Migliavacca, and Peter Pietzuch. 2009. Distributed Complex Event Processing with Query Rewriting. In *DEBS*.

[48] Wil Van Der Aalst. 2011. *Process mining: discovery, conformance and enhancement of business processes*. Springer-Verlag.

[49] Ricardo Vilalta and Sheng Ma. 2002. Predicting Rare Events In Temporal Domains. In *ICDM*.

[50] Di Wang, Elke A. Rundensteiner, and Richard T. Ellison, III. 2011. Active Complex Event Processing over Event Streams. *Proc. VLDB Endow.* 4 (2011), 634–645.

[51] Jie Wang. 2021. An Intuitive Tutorial to Gaussian Processes Regression.

[52] Frans M. J. Willems, Yuri M. Shtarkov, and Tjalling J. Tjalkens. 1995. The context-tree weighting method: basic properties. *IEEE Trans. Information Theory* 41, 3 (1995), 653–664.

[53] Eugene Wu, Yanlei Diao, and Shariq Rizvi. 2006. High Performance Copmlex Event Processing over Streams. In *SIGMOD*.

[54] Zhengdao Xu and Hans-Arno Jacobsen. 2007. Adaptive location constraint processing. In *SIGMOD*.

[55] Zhengdao Xu and Hans-Arno Jacobsen. 2007. Evaluating Proximity Relations Under Uncertainty. In *ICDE*.

[56] Zhengdao Xu and Hans-Arno Jacobsen. 2009. Expressive Location-Based Continuous Query Evaluation with Binary Decision Diagrams. In *ICDE*.

[57] Zhengdao Xu and Hans-Arno Jacobsen. 2010. Processing proximity relations in road networks. In *SIGMOD*.

[58] Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li, Zhili Xiao, Bin Cheng, Jiashu Xing, Yangtao Wang, Tianheng Cheng, Li Liu, Minwei Ran, and Zekang Li. 2019. An End-to-End Automatic Cloud Database Tuning System Using Deep Reinforcement Learning. In *SIGMOD*.

[59] Cheng Zhou, Boris Cule, and Bart Goethals. 2015. A pattern based predictor for event streams. *Expert Syst. Appl.* 42, 23 (2015), 9294–9306.