# Genetic Algorithms for Hyperparameter Optimization in Predictive Business Process Monitoring

Chiara Di Francescomarino[a,*], Marlon Dumas[b], Marco Federici[c,a], Chiara Ghidini[a], Fabrizio Maria Maggi[b], Williams Rizzi[a,d], Luca Simonetto[c,a]

[a]*FBK-IRST, Via Sommarive 18, 38123 Trento, Italy*
[b]*University of Tartu, Ülikooli 18, 50090 Tartu, Estonia*
[c]*University of Amsterdam, 1012 WX Amsterdam, The Netherlands*
[d]*University of Trento, Via Sommarive 9, 38123 Trento, Italy*

## Abstract

Predictive business process monitoring aims at predicting the outcome of ongoing cases of a business process based on past execution traces. A wide range of techniques for this predictive task have been proposed in the literature. It turns out that no single technique, under a default configuration, consistently achieves the best predictive accuracy across all datasets. Thus, the selection and configuration of a technique needs to be done for each dataset. This paper presents a framework for predictive process monitoring that brings together a range of techniques, each with an associated set of hyperparameters. The framework incorporates two automatic hyperparameter optimization algorithms, which given a dataset, select suitable techniques for each step in the framework and configure these techniques with minimal user input. The proposed framework and hyperparameter optimization algorithms have been evaluated on two real-life datasets and compared with state-of-the-art approaches for predictive business process monitoring. The results demonstrate the scalability of the approach and its ability to identify accurate and reliable framework configurations.

*Corresponding author

*Email addresses:* dfmchiara@fbk.eu (Chiara Di Francescomarino), marlon.dumas@ut.ee (Marlon Dumas), federici.marco.94@gmail.com (Marco Federici), ghidini@fbk.eu (Chiara Ghidini), f.m.maggi@ut.ee (Fabrizio Maria Maggi), rizzi.williams@me.com (Williams Rizzi), luca.simonetto.94@gmail.com (Luca Simonetto)

*Keywords:* Predictive Process Monitoring, Hyperparameter Optimization, Genetic Algorithm

## 1. Introduction

*Predictive business process monitoring* is a family of techniques to predict the outcome of ongoing cases of a business process based on logs of past executions of the process. A predictive process monitoring technique allows users to specify a function that labels each completed case of the process into one of multiple possible classes. For example, in a sales process such a function might classify each case into “successful” or “unsuccessful”, while in an issue-to-resolution process, it might classify each case into “customer satisfied” or “customer unsatisfied”. By analyzing the traces of completed cases and their labels, a runtime component (the *monitor* ) continuously estimates the likelihood that the function in question will take a given value (e.g., “successful”) upon completion of each ongoing case of the process.

In previous work [1, 2], we proposed a customizable predictive process monitoring framework comprising a set of techniques to construct models to predict whether or not an ongoing case will ultimately satisfy a given classification function based both on: (i) the sequence of activities executed in the given case, and (ii) the values of data attributes after each activity execution in the case.

The framework can be instantiated by selecting and configuring various machine learning components (clustering and classification techniques). This configurability is an advantage, since previous work in the field has shown that no single technique, with a default configuration, performs well across all datasets [3, 4]. However, configuring such a framework in order to maximize predictive accuracy for a given dataset is non-trivial. For example, in the framework presented in [1, 2], the instantiation of the framework requires one to select among different classification algorithms (e.g., decision trees or random forests) and clustering algorithms (e.g., k-means, agglomerative clustering or dbscan), and to tune the so-called *hyperparameters* (see [3, 4]) upon which these techniques rely. While expert users are able to make suitable choices after significant trial-and-error and tuning, non-experts generally fall back to default choices [5].

A conventional approach to this problem is to perform an exhaustive search over a given set of options and hyperparameter ranges [3]. This means running each possible configuration of the framework on a training dataset, evaluating them on a validation dataset, and comparing the results obtained with different configurations in order to select the most suitable one for the dataset in question. This exhaustive approach, however, on the one hand demands expertise to set the hyperparameter ranges and on the other does not scale up when the space of possible configurations is large. This is often the case since, for example, four possible classification techniques and four clustering techniques, each with three hyperparameters on average, and each hyperparameter with a range of a dozen of possible values, already leads to a space of over 500 possible configurations. A space of this size can take hours or days to be explored exhaustively using a real-life dataset. Another crucial aspect is that, since the suitability of a given configuration can be assessed with respect to multiple goodness measures (e.g., accuracy of the predicted outcomes and earliness of the predictions), different solutions can be the "most suitable" to optimize different measures. Finally, since in the context of predictive business process monitoring two dimensions should be taken into consideration, i.e., control flow and data, it is often the case that a combination of techniques should be employed to retrieve information from both these dimensions. To the best of our knowledge, in the literature, there are no approaches that optimize the selection of techniques and of the corresponding hyperparameter values for addressing multiple combined sub-problems.

In this article, we address the above issues by proposing a predictive process monitoring framework armed with scalable automatic hyperparameter optimization techniques. In particular, we propose a meta-heuristic approach to explore the hyperparameter configuration space based on genetic algorithms. Evolutionary algorithms, like genetic ones, are, indeed, together with grid search, random search and bayesian optimization techniques, among the most used approaches to solve the hyperparameter optimization problem [6]. For instance, they have been successfully applied for optimizing the parameter selection of Support Vector Machines (SVM) [7, 8] and neural network models [9]. Genetic algorithms can either use single-objective functions to optimize a combination of multiple goodness measures, or Pareto fronts to optimize different measures separately. The proposed hyperparameter optimization algorithms, which guarantee the identification of an optimal or near-optimal solution, have been evaluated on two real-life datasets and compared with state-of-the-art approaches for predictive business process monitoring. The results demonstrate the scalability of the approach and its ability to identify accurate and reliable framework configurations.

The approach presented here significantly extends the one presented in a conference version of this article [10]. While in [10], we proposed an exhaustive search approach over user-defined hyperparameter ranges, and a manual approach to compare hyperparameter configurations, in this extended version, we outline more scalable genetic algorithms for hyperparameter optimization and two approaches to aid the user to select the most suitable technique in a given context: one based on single-objective optimization and another based on multi-objective optimization.

The rest of the paper is structured as follows. Section 2 introduces some background elements needed to understand the rest of the paper. Next, Section 3 and Section 5 introduce two motivating scenarios and the overall approach, respectively. The implementation of the approach is then detailed in Section 6. An evaluation is presented in Section 7, while Section 8 and Section 9 conclude the paper with related and future work.

## 2. Background

In this section, we give an overview of the background notions useful in the rest of the paper.

### *2.1. Predictive Business Process Monitoring*

Predictive analytics is a field including techniques to analyze current and historical facts to make predictions about future or otherwise unknown events [11, 12]. *Predictive business process monitoring* [13] aims at applying predictive analytics to business processes. Under the umbrella of the predictive business process monitoring paradigm, different approaches have been proposed and realized [13, 14, 15, 16, 17, 18, 19], motivated by the increasingly pervasive availability of fine-grained event data about business process executions. In particular, in this paper, we focus on approaches that address the problem of *outcome-based predictive process monitoring* [20], which refers to classifying each ongoing case of a process according to a given set of possible outcomes, e.g., “will the customer complain or not?”, “will an order be delivered, cancelled or withdrawn?”. In this paradigm, a user specifies a *business goal* and, based on an analysis of historical cases, she is provided with estimations of the likelihood of achieving the business goal for a given ongoing case. A business goal can be more or less articulated and also cover objectives that need to be achieved by different stakeholders. For example, in a sales process, the business goal can be at two levels: the organization level (e.g., the achievement of a certain profit) and the customer level (e.g., high customer satisfaction).

The predictions generally depend both on: (i) the sequence of activities executed in the given case, and (ii) the values of data attributes after each activity execution in the case. In this paper, we build our contribution on top of the *Predictive Process Monitoring Framework* presented in [1, 2] and described in detail in Section 4. In particular, we start from this existing framework and equip it with a hyperparameter optimization mechanism based on genetic algorithms.

### *2.2. Hyperparameter Optimization*

Traditionally, machine learning techniques are characterized by model parameters and by *hyperparameters* [3, 4]. While model parameters are learned during the training phase so as to fit the data, hyperparameters are set outside the training procedure and used for controlling how flexible the model is in fitting the data. For instance, the number of clusters in the k-means clustering procedure is a hyperparameter of the clustering technique. The impact of hyperparameter values on the accuracy of the predictions can be extremely high. Optimizing their value is therefore important, but, at the same time, optimal values depend on the specific dataset under examination [3, 4].

### *2.3. Genetic Algorithms*

Genetic algorithms [21] are metaheuristic optimization algorithms resembling the natural evolution. By relying on the evolutionary theory of the survival of the fittest and on the ideas of selection and mutation, genetic algorithms aim at simulating the evolution of solutions over different generations so as to eventually identify an optimal or near-optimal solution for an optimization problem. Figure 1 reports the typical workflow followed by a genetic algorithm. An initial set of candidate solutions, i.e., the *initial population*, is first identified, either randomly or according to some heuristics. Therefore, the algorithm evaluates the fitness of each solution and selects the best individuals according to a *fitness function* [21], i.e., a function defining the suitability of the solution to the problem. Then, a set of *genetic operators* such as *crossover* and *mutation* is used to evolve the best individuals and to generate a new population. The new population is, in turn, evaluated and evolved with the same mechanism until a termination condition on the fitness function or on the maximum number of generations is met. The best individuals of the population are therefore identified and returned as a solution of the optimization problem.

*Single- and multi-objective optimization.* An optimization problem can be single-objective, when it involves only one objective function, which can synthesize a combination of different goodness measures, or multi-objective, when more than one objective function has to be optimized. While in the former type of problem, a unique solution exists, which is the solution optimized with respect to the single-objective function, this is not the case for the latter. Indeed, in this case, a set of solutions is found, a.k.a. *Pareto front*. A Pareto front is composed of a set of *non-dominated* solutions, i.e., solutions in which none of the objective functions can be improved in value without degrading some of the other objective values [22].

## 3. Two Motivating Scenarios

We aim at addressing the problem of easing the task of predictive process monitoring, by enabling users to easily select and configure a specific instance of the *Predictive Process Monitoring Framework* to fit a specific dataset. In this section, we introduce two motivating scenarios, which will also be used as a basis for the evaluation of the *Predictive Process Monitoring Framework* tool provided in Section 7.

*Scenario 1. Predicting patient history.* Let *Bob* be a medical director of an oncology department of an important hospital who is interested in predicting the type of medical exams that will be performed on patient *Alice*, and when. In particular, he is interested in knowing if, given the clinical record of Alice: (i) she will need two specific medical exams named *tumor marker CA*_19.9 and *ca* _125 *using meia*, and when, and (ii) it is likely that the execution of a particular medical exam (*CEA_tumor marker using meia*) will be followed by the development of a particular type of allergy in the future. Since

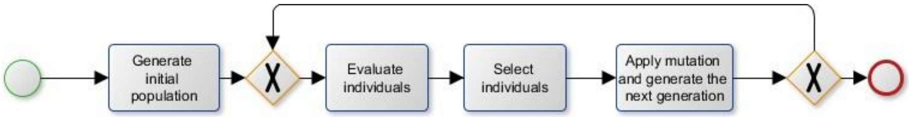

Generate initial population
X
Evaluate individuals
Select individuals
Apply mutation and generate the next generation
X

Figure 1: Genetic algorithm workflow

his department has started an innovative project aiming at using predictive process monitoring techniques to analyze event logs related to the patient history, the hospital records a number of relevant datasets to enable the usage of a predictive process monitoring framework. He can, therefore, use the framework in order to make predictions. However, when ready to use the framework, he finds out that: (i) he needs to specify a list of techniques required to instantiate the framework, and (ii) for each of these techniques, he has to specify a set of hyperparameters. However, being a medical doctor he does not have the necessary knowledge to understand which technique is better to use and the parameters to set, and he does not want to spend time in manually choosing the best ones. His knowledge only enables him to select the business goal he wants to achieve and the dataset of similar cases relevant for the prediction. Thus, an automated way for helping him in understanding which configuration works best for his dataset and specific business goal is needed.

*Scenario 2. Predicting problems in building permit applications.* Let *John* be a clerk handling building permit applications of a Dutch Municipality. The majority of regular building permit applications required for building, modifying or demolishing houses in the Netherlands must be accompanied with the necessary fees and documentation, including design plans, photos and pertinent reports. They are, therefore, often unsuccessfully checked for completeness, and the owner of the application is often asked to send the missing data. This implies extra work from her side and from the building permit application office. Moreover, many of the permit applications also require an environmental license (WABO) and getting the WABO license can either be fast or demand for a long extension of the building permit procedure. This would require a rescheduling of the work of the building permit application office. John is therefore interested in knowing, for example, (i) whether the 4 applications he has just received and of which he has acknowledged the receipt will undergo a series of actions required to retrieve missing data, and (ii) whether these applications will require getting an environmental license, which will entail an extension of the building permit procedure. As in *Scenario 1*, the Municipality where John works stores all the necessary datasets to enable the usage of predictive monitoring techniques, but the difficulty in choosing the right technique and the need of configuring parameters may seriously hamper his ability to use the predictive monitoring tools. Thus, a way for helping him in automatically setting up the correct configuration that works best for his dataset and specific business goal is needed also in this scenario.

## 4. Basic Framework

The basic *Predictive Process Monitoring Framework* , presented in [1, 2], collects a set of machine learning techniques that can be instantiated and used for continuously providing predictions to the user. In detail, the framework takes as input a set of past executions and tries to predict how current ongoing executions will develop in the future. For this purpose, before the process execution, a pre-processing phase is carried out. In this phase, state-of-the-art approaches for clustering and classification are applied to the historical data in order to (i) identify and group historical trace prefixes with a similar control flow, i.e., to delimit the search space on the basis of the control flow (control flow-based clustering), and (ii) get a precise classification based on data of traces with similar control flow (a data-based classifier is built for each cluster).

The rationale for separating the problem of predictive business process monitoring into two sub-problems (a clustering sub-problem and a classification sub-problem) is that, in predictive business process monitoring, we have to deal with the so-called complex symbolic sequences [23]. A complex symbolic sequence is a sequence consisting of events (with certain labels), each of which is associated with a payload consisting of attribute-value pairs. This structure can be seen as two equal-length sequences (the sequence of event labels, and the sequence of event payloads). In addition, since complex symbolic sequences represent the executions of a business process, they can be of very different lengths: some sequences are very short (e.g., process executions that finish after a handful of events because they are aborted) and some are very long (e.g., process executions where there is a lot of rework and exception handling). The above two characteristics lead us to rely on a two-phased cluster-and-classify approach, where the clustering phase is used to handle the sequence of event labels (by grouping them into clusters of similar prefixes), and the classification phase is used to handle the event payloads.

Clusters and classifiers computed as outlined above are stored and used at runtime to classify new traces during their execution. In particular, a given trace prefix is matched to a cluster, and the corresponding classifier is used to estimate the (class) probability for the trace to achieve a certain

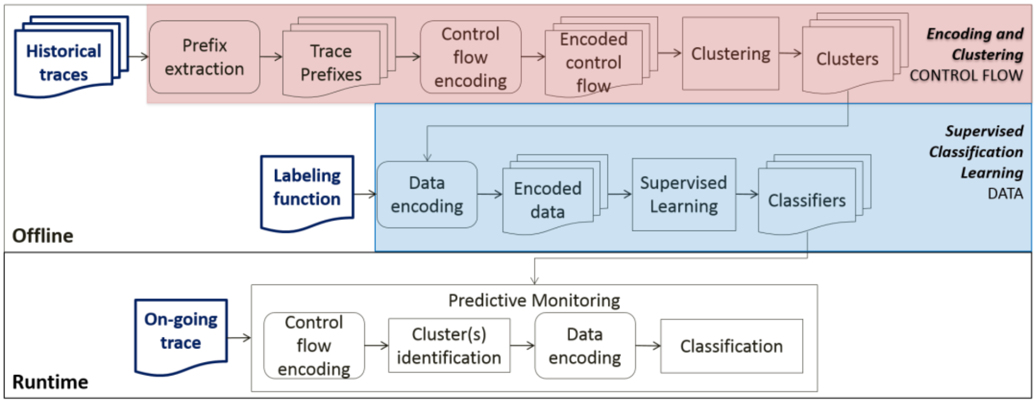


Figure 2: *Predictive Process Monitoring Framework*

outcome and the corresponding (class) support (that also gives a measure of the reliability of the classification algorithm outcomes).

The proposed approach is akin to ensemble methods, where multiple classifiers are constructed to address a given classification problem. Indeed, in our approach, we construct one classifier per cluster. However, the proposed approach does not qualify as an ensemble method as it does not involve any voting nor error-correction mechanism to combine the predictions made by each classifier in the ensemble. Instead, the proposed approach is a non-ensemble multiple-classifier approach, where a given sample (trace prefix) is routed to a single classifier among multiple ones. The justification for this approach is that the trace prefixes for which we need to make predictions are highly heterogeneous: there is a large variability in the lengths of the trace prefixes (some consist of 1-2 events, others may consist of over a dozen events) and the alphabet of event labels varies considerably across trace prefixes. By applying a clustering approach, we divide the set of trace prefixes into more homogeneous clusters, so that we can build simpler and potentially more accurate classifiers for each cluster. The overall picture of the framework is illustrated in Figure 2.

Within such a framework, we can identify three main modules: the *encoding*, the *clustering* and the *supervised classification learning* module. Each of them can be instantiated with different techniques. Figure 3 shows possible instances of the framework.[1]

[1] Note that the techniques mentioned in Figure 3 and used in our experiments have

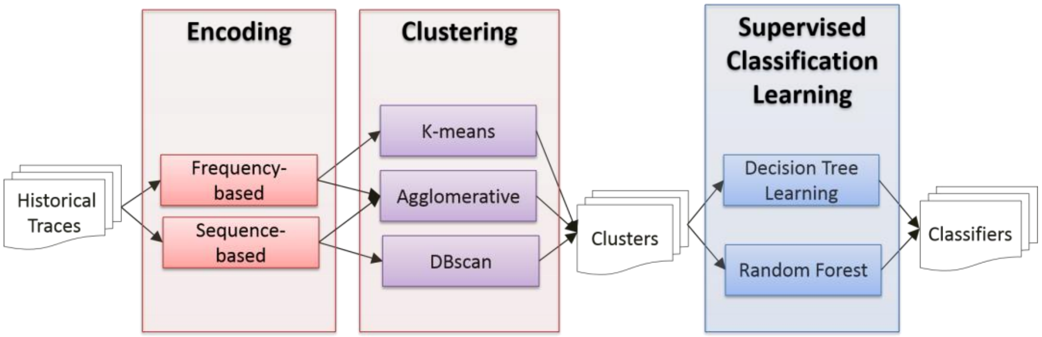


Figure 3: Framework instances overview

Examples of encodings are *frequency-based* (a.k.a. term-frequency [1, 24]) and *sequence-based* [1, 25]. The former is realized representing a trace as a vector of event occurrences (on the alphabet of the events), while, in the latter, the trace is encoded as a sequence of events. These encodings can then be passed to a clustering technique. Examples of these techniques are *dbscan clustering* [26], *k-means clustering* [27] and *agglomerative clustering* [28]. The *Euclidean* distance, used by k-means, is computed starting from the frequency-based encoding, while the *edit* distance, used by dbscan, is computed starting from the sequence-based encoding of the traces. The supervised learning module can include learning techniques like *decision tree* and *random forest*.

Each of these techniques requires a number of hyperparameters (specific for the technique) to be configured. For example, k-means and agglomerative clustering take as input the number of clusters, while dbscan requires the minimum number of points in a cluster and the minimum cluster ray. In addition, the framework also requires the configuration of other parameters, such as:

- *length of prefixes* of historical traces to be grouped in clusters and used for training the classifiers;

been chosen based on the characteristics of the available datasets (which do not contain a particularly large amount of data) and are purely exemplificative. This still allows the user to use any other machine learning technique to solve the clustering and classification tasks. Indeed, the hyperparameter optimization with genetic algorithms is not tailored to a specific technique but it is general and can be applied to any combination of techniques and hyperparameters.

- a *voting mechanism* [29], so that the *p* clusters closest to the current trace are selected, the prediction according to the corresponding classifiers estimated, and the prediction with the highest number of votes (from the classifiers) returned;
- when the prediction is related to the completion time interval, a *mechanism for the definition of the time intervals* (e.g., *q* intervals of the same duration, based on *q*-quantiles, or based on a normal distribution of the time).

The framework can then be instantiated through different combinations of these techniques and hyperparameters. Although, in the pre-processing phase, the intermediate results are stored for reuse, different configurations can demand for different intermediate results. Each choice of technique (and hyperparameters) in a configuration can indeed affect the *Predictive Process Monitoring Framework* flow at different stages. For instance, the choice of the encoding type affects the clusters built from the historical traces; the choice of the classification learning technique does not affect the clusters but it does affect the classifiers built on top of them.

The framework has been implemented as an Operational Support (OS) provider of the OS Service 2.0 [30, 31] of the ProM toolset. In particular, the OS service is able to interact with external workflow engines by receiving at runtime streams of events and processing them through the providers.

## 5. Genetic-Enhanced Framework

The *Genetic-Enhanced Predictive Process Monitoring Framework* proposed in this work relies on arming the basic *Predictive Process Monitoring Framework* with genetic algorithms to support users in the choice of the framework configuration that best suits their dataset and business goal. Indeed, genetic algorithms allow for the exploration of the search space of the different framework instances, by starting from an initial population and evolving it while optimizing a fitness function. For this purpose, the approach proposed in this work adds to the basic *Predictive Process Monitoring Framework* two levels of abstraction and aggregation: one aiming at providing an evaluation of each configuration on a validation set of traces and a second one for evolving a population of configurations in order to identify the best configuration(s). These levels are implemented in the *Genetic-enhanced Tuner*. Therefore, the *Genetic-Enhanced Predictive Process Monitoring Framework* is the combination of the basic *Predictive Process Monitoring Framework* and the *Genetic-Enhanced Tuner*.

### *5.1. Architecture of the Genetic-Enhanced Tuner*

Figure 4 shows the architecture of the *Genetic-Enhanced Tuner*. The tuner takes as inputs from the user a training set, a business goal, and a validation set and returns the configuration(s) that best suit the specific dataset and business goal.

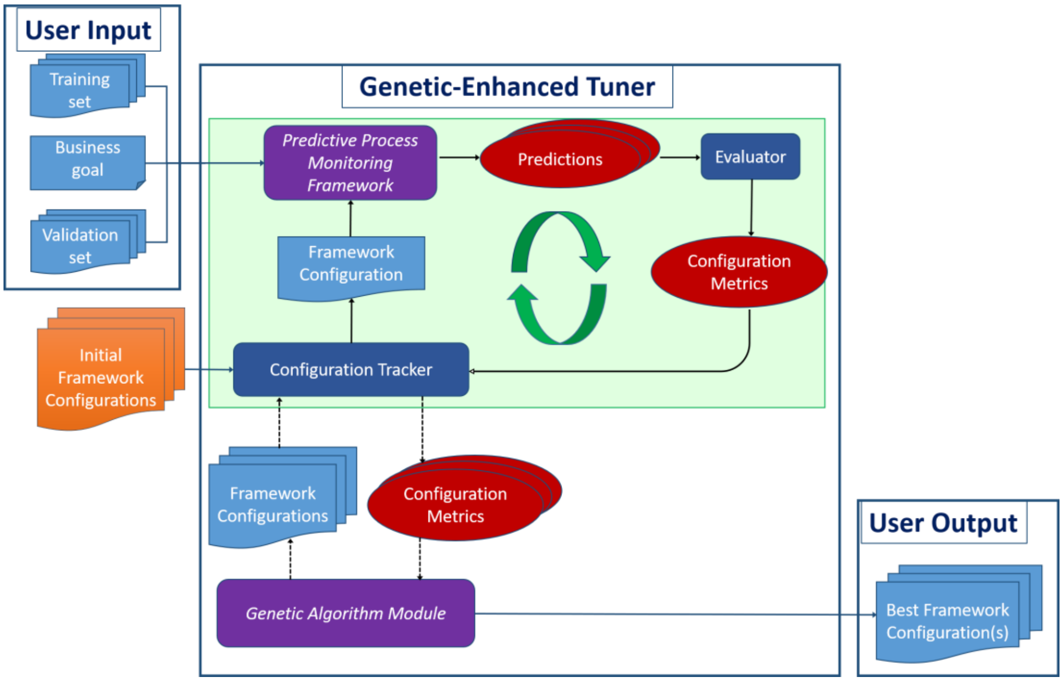
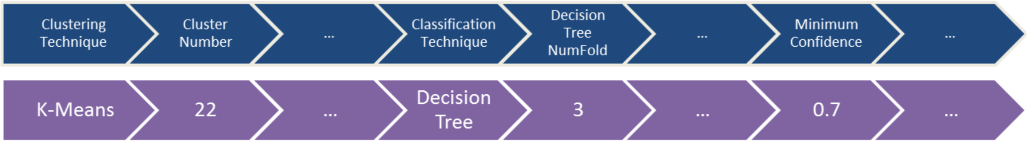


Figure 4: Internal architecture of the Genetic-Enhanced Tuner

The flow of the logical architecture starts with a set of initial (randomly generated) framework configurations (*initial population*), which are sent to the *Predictive Process Monitoring Framework* in charge of producing a set of predictions for each configuration. The *Configuration Tracker* encodes each framework configuration in the right format for the *Predictive Process Monitoring Framework* that will generate the predictions for the traces in the validation set. An *Evaluator* will then be in charge of evaluating the provided predictions according to a set of fitness functions defined in terms of *configuration metrics*. A configuration metric is a measure of the goodness of a given configuration. Three such measures of goodness are discussed below.

As they are computed, the configuration metrics are sent to the *Configuration Tracker* that takes care of collecting them and providing them to the *Genetic Algorithm Module*. The *Genetic Algorithm Module*, in turn, uses the configuration metrics in order to compute a single- or a multi-objective *fitness function* and rank the configurations according to this function. For the *Genetic Algorithm Module*, each framework configuration is an individual of the population which is encoded as a vector where every element represents a configuration parameter. For instance, Figure 5 gives a hint of a possible encoding of a framework configuration. From the set of configurations, the module selects the best ones, which are used, by applying mutation operators, to produce the next generation of framework configurations.

The new population of configurations is sent again to the *Configuration Tracker* and, one by one, to the *Predictive Process Monitoring Framework* for the evaluation. The process is iterated, until a threshold is reached in terms of number of generations or number of generations without any improvement in the fitness function (a.k.a. stagnation value). Once the genetic algorithm execution is terminated, one or more framework configurations are returned to the user as output, based on whether a single- or a multi-objective approach to the optimization problem is taken. These configurations can be used to tune the *Predictive Process Monitoring Framework* and run it on a testing set.



| Clustering Technique | Cluster Number | ... | Classification Technique | Decision Tree NumFold | ... | Minimum Confidence | ... |
|---|---|---|---|---|---|---|---|
| K-Means | 22 | ... | Decision Tree | 3 | ... | 0.7 | ... |

Figure 5: Example of encoding of a framework configuration

### 5.2. Configuration Metrics

To instantiate the above architecture, we need to define metrics to evaluate the quality of the predictions produced by a given configuration. Here, we define three metrics corresponding to an online monitoring scenario in which a prediction is made for a given case, as soon as the class probability returned by the classifier is above a given user-defined threshold. In other words, every time an event occurs in a given case, we convert the current trace prefix into a feature vector, and we feed this feature vector to the classifier associated to the closest cluster relative to this trace prefix. If the class probability returned by the classifier is above the threshold, we return the predicted class and no further predictions are made for this case (i.e., the case is considered to have been "classified"). If the class probability is below the threshold, no prediction is made at this stage. Sometimes, a case might finish before a prediction is made. In this case, we say that there is a *prediction failure*.

Given the above, one possible measure of accuracy is the ratio between the number of times a correct prediction is made and the total number of predictions made. However, since in some cases no prediction is made at all, we need to complement this metric with a *failure-rate*. Finally, since an early prediction is better than a late prediction, we also define a third metric, namely the *earliness*, which captures how early in a given case the prediction is made. More in detail, these metrics are defined as follows.

- **Accuracy.** This metric is defined with respect to a *gold standard* that indicates the correct labeling of each trace. In our experiments, we extracted the gold standard by evaluating the achievement of a business goal in each completed trace in the testing set. Given the gold standard, we classify predictions made at runtime into four categories: *i*) true-positive ($T_P$ : positive outcomes correctly predicted); *ii*) false-positive ($F_P$ : negative outcomes predicted as positive); *iii*) true-negative ($T_N$ : negative outcomes correctly predicted); *iv*) false-negative ($F_N$ : positive outcomes predicted as negative). Accuracy indicates how many times a prediction was correct.

- **Failure-rate.** During the replay of each trace in the testing log, we give a prediction every $X$ events (starting from the first event in the trace). While replaying a trace, we consider a prediction reliable and we stop the replay when the corresponding class probability is above the given minimum class probability threshold. Therefore, it can happen that, when replaying a trace of the testing set, the end of the trace is reached and no prediction has been made. In this case, the answer of the predictor is "maybe" to indicate that it was not possible to provide a reliable prediction. The percentage of traces in the log that lead to a failure in the prediction is called failure-rate [32].

- **Earliness.** The earliness of the prediction [1] is defined as one minus the ratio between the index indicating the position of the last evaluation point (the one corresponding to the reliable prediction) and the length of the trace under examination. The earliness of a prediction is directly dependent on the chosen class probability threshold. The lower the class probability threshold is, the earlier a prediction is given.

Note that the proposed framework leaves us the flexibility to choose which one of the three metrics should be preferred based on different scenarios. For instance, we can favor the accuracy over the failure-rate and the earliness (by choosing a high class probability threshold) or we can favor a low failure-rate to the detriment of the accuracy (by specifying a low class probability threshold).

An alternative approach would have been to fix a given prefix length (say 3) and to make predictions when a trace reaches this length. We could then measure the Area Under the Curve (AUC) of these predictions. This could then be repeated for all possible prefix lengths, and a mean of the AUC values (across prefix lengths) could be used as an accuracy measure for a given configuration. This approach, however, does not give us a direct idea of how early a prediction can be made during the execution of a case.

### *5.3. Genetic Algorithm Module*

In this work, the choice of the single-objective fitness function is mainly based on the assumption that a generic user of the *Predictive Process Monitoring Framework* would likely be interested in balancing the correctness of the returned predictions (accuracy) and the rate of the non-returned predictions (failure-rate). For instance, John, in *Scenario 2* of Section 3, will likely be interested in getting as many predictions as possible, while preserving an adequate level of correctness. Moreover, we also include earliness for promoting early predictions, although with a lower weight.

Given the above, we define the single-objective fitness function as:[2]

$$\mathit{fitness} = \mathit{accuracy} + (1.0 - \mathit{failureRate}) + (\mathit{earliness}/\mathit{reducingFactor})$$

Differently from the single-objective approach, the multi-objective approach offers more freedom to the user, leaving her the possibility to choose

[2] reducingFactor is a value used to limit the influence of the earliness in the fitness calculation. In this work, the reducing factor has been set to 20.

the most suitable solution from a set of Pareto optimal solutions. For example, in a hospital scenario, a doctor who relies on a prediction based on historical data to make a decision about a treatment for a patient can choose a treatment that favors the highest accuracy, the one with the lowest failure-rate or the one that incorporates the best balance between the two, according to her specific needs. The higher flexibility of the multi-objective approach is guaranteed at the cost of possibly lower performance or longer computation times with respect to the single-objective approach. The fitness function investigated in this case is calculated as:

$$score = [accuracy, (1.0 - failureRate)]$$

Also in this case, the choice of the fitness function is based on the assumption that a user would mainly be interested in the interplay between accuracy and failure-rate. For instance, Bob, in *Scenario 1* of Section 3, would be interested in choosing among different configurations, so as to be able to (i) get only accurate predictions (highest accuracy), in case of highly critical situations; (ii) always get predictions, even if they could be incorrect (lowest failure-rate), in case he just needs hints on what to do next; (iii) get predictions balancing high accuracy and low failure-rate.

Different types of genetic algorithms for the multi-objective optimization exist. In this work, we chose to use the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [33]. Indeed, this is a well known and adopted algorithm, it is easy to understand and implemented in many libraries. Moreover, it has been used for solving problems of hyperparameter optimization [34] and it has the same or better performance than its main competitor, the Strength Pareto Evolutionary Algorithm II (SPEA-II) [35].

## 6. Implementation

In order to realize the approach described in Section 5, the *Genetic Algorithm Module*, depicted in Figure 4, has been implemented by instantiating two existing genetic algorithm frameworks, one supporting the single-objective approach and the other supporting the multi-objective approach. Moreover, the *Configuration Tracker* module has been implemented to interface the *Genetic Algorithm Module* with the *Predictive Process Monitoring Framework*. All modules have been implemented in Java.

### *6.1. Single-Objective Approach*

The implementation used for the single-objective approach is based on the Watchmaker Framework,[3] a simple-to-use library that provides several features ready to be used in a custom-made genetic algorithm.

*Framework.* The Watchmaker Framework is an optimized library for genetic algorithms written in Java and using a well documented and non-invasive API. It supports multi-threading and provides several genetic operators, evolution schemes and termination conditions.

*Framework instantiation.* The Watchmaker Framework has been instantiated by implementing the classes of the library reflecting the typical components of a genetic algorithm, as reported in the following.

- **Encoding.** A predictive monitoring framework configuration is encoded as a vector where every element represents a framework parameter.

- **Operators.** The uniform crossover and uniform mutation have been implemented as extensions of the crossover and mutation abstract classes of the framework. The sigma scaling technique available in the framework has been used for the selection of the best individuals.

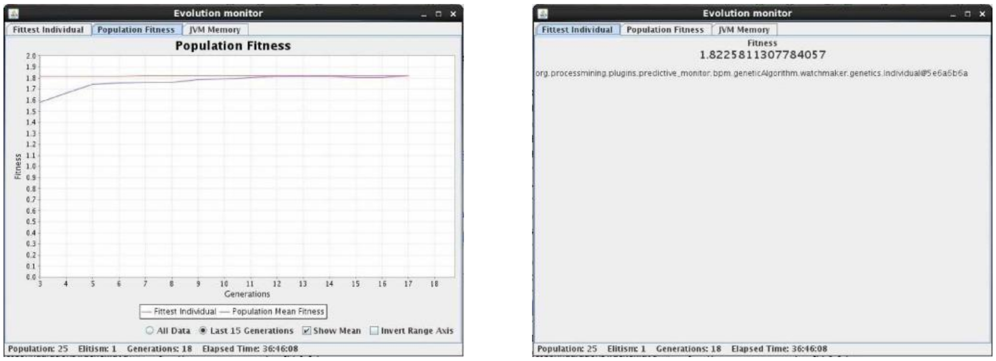
- **Fitness function.** As reported in Section 5, the fitness function has been set to:

  $$fitness = accuracy + (1.0 - failureRate) + (earliness/reducingFactor)$$

*Output.* The output generated by this approach consists of a single solution (framework configuration) with the best fitness in the last generation. This solution should approximate the framework configuration that better fits the problem.

*Custom evolution monitor GUI.* The library has a GUI that can show important statistics about population, fittest individual, time elapsed, memory usage. The GUI has been adapted to be able to show the generation progress and the best individual fitness. Figure 6a and 6b report the screenshots of

[3] http://watchmaker.uncommons.org/



Population: 25 Elitism: 1 Generations: 18 Elapsed Time: 36:46:08

(a) Best and mean generation fitness trend across the last 15 iterations

```
Fitness
1.8225811307784057
org.processmining.plugins.predictive_monitor.bpm.geneticAlgorithm.watchmaker.genetics.individual@5e6a6b6a
```

Population: 25 Elitism: 1 Generations: 18 Elapsed Time: 36:46:08

(b) Fittest individual of the last generation

Figure 6: Customized evolution monitor GUI

two of the panels of the customized GUI showing the trend of the best and mean fitness per generation (6a) and the fitness value of the fittest individual (6b), respectively.

### *6.2. Multi-Objective Approach*

The framework instantiated for the multi-objective approach is Jenes 2.0.

*Framework.* The Jenes 2.0 framework[4] is an easy-to-use library with multi-threading support, efficient memory management, multiple genetic operators and a variety of different algorithms.

*Framework instantiation.* Similarly to the Watchmaker Framework, also Jenes has been instantiated by implementing the classes reflecting the typical components of a genetic algorithm, as reported in the following.

- **Encoding.** A predictive monitoring framework configuration is encoded as a vector where every element represents a framework parameter.
- **Operators.** As this implementation uses an encoding of the individual that fits with the needs of the faced problem, the existing genetic

[4] The Jenes 2.0 framework has been developed by Intelligentia s.r.l and the Computational and Intelligent System Engineering Lab (CISELab) at the University of Sannio.

operators, crossover and mutation have been used without any custom adaptation. In particular, the tournament selection, the two-point crossover and the uniform mutation have been used for the selection of the best individuals, crossover and mutation, respectively.

- **Fitness function.** A fitness class provides a measure for evaluating the score of each individual. As already reported in Section 5, the fitness class has been instantiated with the following score value:

  $$score = [accuracy, (1.0 - failureRate)]$$

  The score is an array with two values, so that a Pareto front can be obtained by the execution of the algorithm.

*Output.* The output of this approach is a set of solutions (framework configurations) corresponding to the Pareto front of the last generation of solutions.

## 7. Evaluation

This section reports an evaluation of the *Genetic-Enhanced Predictive Process Monitoring Framework* using both the single- and the multi-objective approach.

### *7.1. Research Questions*

In order to evaluate the proposed approaches, we aim at answering the following two research questions:

**RQ1** *How good is the framework configuration obtained by using a single-objective genetic algorithm approach?*
**RQ2** *How good are the framework configurations obtained by using a multi-objective genetic algorithm approach?*

**RQ1** focuses on the single-objective approach and aims at investigating whether it is able to return a configuration which provides accurate predictions with a low failure-rate. **RQ2** focuses on the multi-objective approach and aims at investigating whether it is able to return a set of configurations that are the most suitable for optimizing accuracy and failure-rate separately. In particular, to answer **RQ1**, the following three research questions have been investigated:

**RQ1.1** *What is the shape of the fitness function?*
**RQ1.2** *How good are the results obtained by using the configuration returned by the algorithm with respect to the ones obtained with state-of-the-art predictive process monitoring approaches?*
**RQ1.3** *How efficient is the algorithm in returning the suggested configuration?*

**RQ1.1** aims at analyzing the fitness function shape in order to understand how fast the function improves across the algorithm generations. **RQ1.2** aims at evaluating whether the configuration provided by our fully automatic hyperparameter optimization approach generates predictions with comparable accuracy and failure-rate with respect to the ones obtained by applying state-of-the-art predictive process monitoring approaches. Finally, **RQ1.3** investigates how much time is required by the algorithm to find the best configuration.

Symmetrically, to answer **RQ2**, the following three research questions have been investigated:

**RQ2.1** *What is the shape of the Pareto front?*
**RQ2.2** *How good are the results obtained by using the configurations returned by the algorithm with respect to the ones obtained with state-of-the-art predictive process monitoring approaches?*
**RQ2.3** *How efficient is the algorithm in returning the suggested set of configurations?*

**RQ2.1** focuses on evaluating the number and the distribution of the configurations on the Pareto front. **RQ2.2** aims at evaluating whether the configurations automatically identified by the algorithm generate predictions optimal in at least one of two metrics – accuracy and failure-rate. Finally, **RQ2.3** investigates how much time is required by the algorithm to get the suggested set of configurations.

### *7.2. Datasets*

For the evaluation of the proposed approaches, we used two datasets provided for the BPI Challenges 2011 [36] and 2015 [37], respectively. The first dataset consists of an event log pertaining to the treatment of patients diagnosed with cancer in a Dutch academic hospital. The log contains 1,140 traces and 149,730 events referring to 623 different activities.

In our experiments, we formulated business goals in terms of Linear Temporal Logic (LTL) formulas. LTL [38] is a modal logic with modalities devoted to describe time aspects. Classically, LTL is defined for infinite traces. However, when focusing on the compliance of business processes, we use a variant of LTL defined for finite traces (since business processes are supposed to complete eventually). The business goals investigated with the first dataset are reported below as LTL formulas:

$$\phi_{11} = \mathbf{F}(\text{“}\mathit{tumor\ marker\ CA} - 19 - 9\text{”}) \vee \mathbf{F}(\text{“}\mathit{ca} - 125\ \mathit{using\ meia}\text{”})$$

$$\phi_{12} = \mathbf{G}(\text{“}\mathit{CEA} - \mathit{tumor\ marker\ using\ meia}\text{”} \rightarrow \mathbf{F}(\text{“}\mathit{squamous\ cell\ carcinoma\ using\ eia}\text{”}))$$

The first formula assesses that the medical exam for measuring the tumor marker CA-19-9 or the medical exam for measuring the tumor marker ca-125 using meia will eventually occur, while the second one states that it always happens that the occurrence of the medical exam CEA-tumor marker using meia will eventually be followed by the medical exam for the diagnosis of squamous cell carcinoma.

The second dataset was provided for the BPI Challenge 2015 by a Dutch Municipality. It is composed of 1,199 traces and 52,217 events referring to 398 activities. The data contains all building permit applications over a period of approximately four years. In this case, the business goals to be analyzed are defined by the following LTL formulas:

$$\phi_{21} = \mathbf{F}(\text{“}\mathit{start\ WABO\ procedure}\text{”}) \wedge \mathbf{F}(\text{“}\mathit{extend\ procedure\ term}\text{”})$$

$$\phi_{22} = \mathbf{G}(\text{“}\mathit{send\ confirmation\ receipt}\text{”} \rightarrow \mathbf{F}(\text{“}\mathit{retrieve\ missing\ data}\text{”}))$$

The first formula assesses that the WABO procedure will eventually start and that the procedure term will eventually be extended, while the second one states that it always happens that sending the confirmation receipt is eventually followed by the retrieval of missing data.

*7.3. Experimental Setting and Procedure*

To evaluate the goodness of the proposed approaches, we proceeded as follows:

- we labeled each trace of the datasets according to whether it is compliant or not with the specified formulas; [5]

[5] We used an automated LTL checker for the labeling. In general, the framework works with any type of labeling, including manual categorizations of the traces.

- we divided the datasets into three different parts: (i) the training set, consisting of 70% of the whole dataset, (ii) the validation set accounting for 20% of the dataset, and (iii) the testing set, composed of the remaining 10%;
- for each algorithm run, we used the training and validation sets to perform the selection of the best framework configuration(s);
- after having identified the best configuration(s), we evaluated them using the testing set to measure their performance on unseen data.

The genetic algorithms used in the two approaches have themselves a set of parameters (meta-hyperparameters). For the single-objective approach, we set the parameters of the genetic algorithm as follows:

- **Population size** of 25 individuals; the choice has been taken as a good trade-off between quality of the results and computation time.
- **Mutation rate** of 0.02, i.e., in the mutation stage only 2% of the resulting individuals have the probability to be mutated. A too low mutation rate would render this stage useless, and a too high mutation rate would be counterproductive to the approach.
- **Elitism** of 1; elitism is used to speed up the convergence of the algorithm. Setting the elitism value to 1 means that the individuals with the highest fitness score would bypass all the intermediate algorithm stages and go straight to the next generation.
- **Stagnation value** of 10; stagnation value represents the number of generations without any improvement in the fitness function that would result in the termination of the algorithm. The chosen value should allow the algorithm to have enough time to search for better solutions.

For the multi-objective approach, the genetic algorithm was configured as follows:

- **Population size** of 25 individuals, like in the case of the single-objective approach. Relying on the same amount of individuals, indeed, helps the comparison between the two approaches.

- **Mutation rate** of 0.02; this value is also the same as the one chosen for the single-objective approach.

- **Crossover probability** of 0.8; this is the probability that two parents are chosen to create offsprings.

- **Epochs** of 15; differently from the single-objective approach, in this case, a maximum number of generations is used for terminating the algorithm, rather than a stagnation value. As for the population size, this value has been chosen as a good tradeoff between quality of the results and computation time.

### *7.4. Experimental Results*

Below, we report the results of the evaluation for each of the two approaches. The results are based on four runs (i.e., the approaches have been applied to each of the two datasets and for each of the two business goals, as described in Section 7.2).

### *7.4.1. Single-Objective Approach Experimental Results*

In order to answer the first research question (**RQ1.1**), we looked at the trend of the fitness function for different generation numbers. Figures 7 and 8 show the values of the fitness function corresponding to the four business goals (formulas $\phi_{11}$, $\phi_{12}$, $\phi_{21}$, and $\phi_{22}$). The y axis reports the values of the fitness function,[6] while the x axis reports the generation number. The blue line shows the value of the best fitness, the red one the value of the mean fitness.

By analyzing the plots, first, we can observe that the best fitness grows over time up to a given generation, indicating that the algorithm progresses towards an improvement of the fitness function. Second, we note that there is a substantially different trend between the runs executed on the first dataset and the ones executed on the second dataset. The plots corresponding to the first two business goals ($\phi_{11}$ and $\phi_{12}$) show that the best fitness starts from an initial value of around 80% of the maximum possible value and it rapidly reaches a value of around 90% of the maximum value, so that the next improvements are only marginal. The other two runs provide, instead, a

[6] The minimum value for the y axis has been chosen to be 1 instead of 0, in order to improve the readability of the plot.

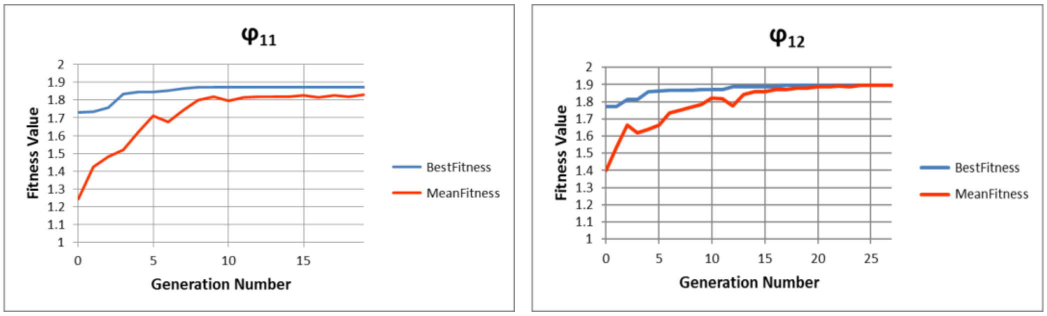


Figure 7: Fitness function trend for the BPI Challenge 2011 dataset

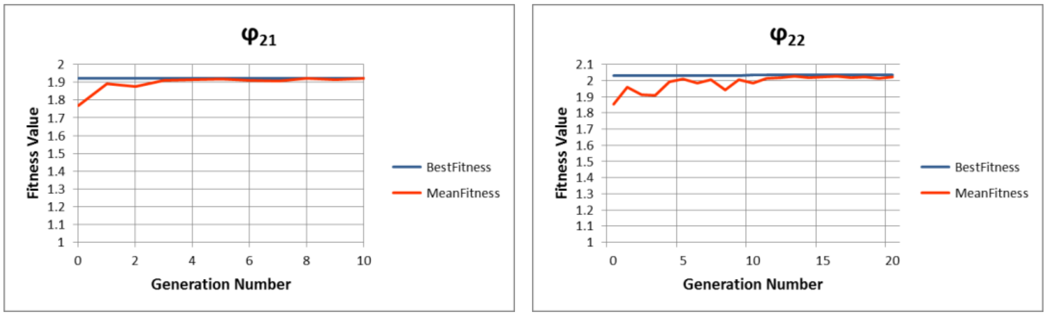


Figure 8: Fitness function trend for the BPI Challenge 2015 dataset

flatter plot where the best fitness is reached already at the initial generations. In particular, in the plot of function $\phi_{21}$, we can see that the number of total generations is 11, indicating that the best fitness was already found in the first generation (the stagnation value for this experiment was set to 10). The plot related to $\phi_{22}$ reveals a higher number of generations, although the plot is still quite flat, indicating only some marginal improvement over time. The mean fitness follows the classical trend for both datasets: it starts with a score lower than the best fitness and gradually reaches it, though with some fluctuations.

From this analysis, we conclude that the selection of the configuration is less critical for the second dataset than for the first one, since for the second dataset an initial population of random configurations already leads to the highest fitness. Overall, we can answer **RQ1.1** by noting that the evolution of the fitness depends on the faced business goal and dataset, though for the considered datasets and business goals it converges in a small number of generations.

| Rule | Validation Fitness | Testing Fitness | Fitness Difference | Metrics | Validation Results | Testing Results | Metrics Difference |
|---|---|---|---|---|---|---|---|
| $\phi_{11}$ | 1.8809 | 1.6952 | -0.1858 | Accuracy | 0.8649 | 0.7264 | -0.1385 |
| | | | | Failure-Rate | 0.0263 | 0.0702 | +0.0439 |
| | | | | Earliness | 0.8477 | 0.7787 | -0.069 |
| $\phi_{12}$ | 1.8947 | 1.8689 | -0.026 | Accuracy | 0.8465 | 0.8393 | -0.0072 |
| | | | | Failure-Rate | 0 | 0.0175 | +0.0175 |
| | | | | Earliness | 0.9641 | 0.9385 | -0.0256 |
| $\phi_{21}$ | 1.9203 | 1.9591 | +0.0388 | Accuracy | 0.8703 | 0.9091 | +0.0388 |
| | | | | Failure-Rate | 0 | 0 | 0 |
| | | | | Earliness | 1 | 1 | 0 |
| $\phi_{22}$ | 2.0336 | 1.9951 | -0.0384 | Accuracy | 0.9874 | 0.9504 | -0.037 |
| | | | | Failure-Rate | 0 | 0 | 0 |
| | | | | Earliness | 0.9234 | 0.8947 | -0.0287 |

Table 1: Results obtained with the validation and testing sets using the single-objective approach

In order to answer **RQ1.2**, we used the configuration computed by optimizing the fitness function (i.e., the one obtained by balancing accuracy and failure-rate on a validation set) on a set of new traces, i.e., the testing set. Table 1 reports, for each dataset and business goal, the values of the fitness function of the configuration suggested by the algorithm as the best one, i.e., the one with the highest fitness in the tuning phase (column *Validation Fitness*), as well as the values of the fitness function obtained using the testing set (column *Testing Fitness*). Moreover, the table details the values of the configuration metrics used for computing the fitness function obtained with the best configuration on the validation set (*Validation Results* column) and on the testing set (*Testing Results* column).

Looking at the results, we observe some salient patterns. The testing fitness values are close to the corresponding validation fitness values: this means that the solutions found by the single-objective approach are not overfitting the validation sets. Indeed, by looking at the values of the fitness function in the validation and in the testing set, we observe that, when the configuration is tested on unseen traces, despite a slightly higher decrease in the value of the fitness function in the case of $\phi_{11}$, the difference between the values of tuning and testing fitness function is overall quite low (in the case of $\phi_{12}$, $\phi_{21}$, and $\phi_{22}$ the difference is of at most 0.038). By looking more in detail at the specific configuration metrics, we observe a similar pattern: between the tuning and the testing phase, we notice an increase in the accuracy for $\phi_{21}$, a slight decrease for $\phi_{12}$ and $\phi_{22}$ and a more significant decrease (of around 0.14) for $\phi_{11}$. For all the four business goals, the failure-rate shows an in-

| Rule | Accuracy | Failure-Rate |
|---|---|---|
| $\phi_{11}$ | 0.875 | 0.5789 |
| $\phi_{12}$ | 0.8421 | 0.6667 |
| $\phi_{21}$ | 0.9506 | 0.3306 |
| $\phi_{22}$ | 0.9829 | 0.0413 |

Table 2: Results obtained with the approach in [13] with manually selected hyperparameters

| Rule | Objective | Metrics | Validation Results | Testing Results | Metrics Difference |
|---|---|---|---|---|---|
| $\phi_{11}$ | Accuracy | Accuracy<br>Failure-Rate | 0.9187<br>0.4605 | 0.86<br>0.5614 | -0.0587<br>0.1001 |
| | Failure-Rate | Accuracy<br>Failure-Rate | 0.6009<br>0 | 0.86<br>0 | 0.2591<br>0 |
| | Balance | Accuracy<br>Failure-Rate | 0.8457<br>0.1754 | 0.8353<br>0.2544 | -0.0104<br>0.079 |
| $\phi_{12}$ | Accuracy | Accuracy<br>Failure-Rate | 0.8931<br>0.4254 | 0.9516<br>0.4561 | 0.0585<br>0.0307 |
| | Failure-Rate | Accuracy<br>Failure-Rate | 0.75<br>0 | 0.7321<br>0.0175 | -0.0179<br>0.0175 |
| | Balance | Accuracy<br>Failure-Rate | 0.7704<br>0.0877 | 0.8716<br>0.0439 | 0.1012<br>-0.0438 |
| $\phi_{21}$ | Accuracy | Accuracy<br>Failure-Rate | 0.907<br>0.2803 | 0.9159<br>0.1157 | 0.0089<br>-0.1646 |
| | Failure-Rate | Accuracy<br>Failure-Rate | 0.8703<br>0 | 0.9091<br>0 | 0.0388<br>0 |
| | Balance | Accuracy<br>Failure-rate | 0.8703<br>0 | 0.9091<br>0 | 0.0388<br>0 |
| $\phi_{22}$ | Accuracy | Accuracy<br>Failure-Rate | 1<br>0.1171 | 1<br>0.2562 | 0<br>0.1391 |
| | Failure-Rate | Accuracy<br>Failure-Rate | 0.9791<br>0 | 1<br>0 | 0.0209<br>0 |
| | Balance | Accuracy<br>Failure-Rate | 0.9871<br>0.0293 | 0.9561<br>0.0579 | -0.031<br>0.0286 |

Table 3: Results reported in [10] related to the semi-automatic optimization of the hyperparameters

crease of at most 0.044 between the tuning and the testing phase. Finally, the earliness score decreases of at most 0.07 from the tuning to the testing phase. These considerations indicate that the single-objective approach is quite stable across different runs and allows us to effectively find framework configurations that give accurate predictions in all executions.

By relying on the assumption that a generic user of the *Genetic-Enhanced Predictive Process Monitoring Framework* is likely interested in balancing the correctness of the returned predictions and the rate of the non-returned predictions, we compared the quality of the obtained configurations in terms of accuracy and failure-rate with two state-of-the-art techniques in the context of the predictive business process monitoring: the on-the-fly approach presented in [13] and the semi-automatic approach investigated in [10].

The on-the-fly approach [13] builds a classification model on-the-fly in order to provide predictions about the achievement of a business goal in an execution trace. In particular, at runtime, the approach identifies trace prefixes of historical cases with a control flow close to the one of the ongoing trace (i.e., trace prefixes that have an edit distance from the current prefix lower than a given similarity threshold) and uses these similar prefixes to build a decision tree on-the-fly. In other terms, differently from the *Genetic-Enhanced Predictive Process Monitoring Framework* used in this work, no clustering phase is carried out, but only a classification technique - decision tree learning - is used and its hyperparameters are manually tuned.

The semi-automatic approach [10], instead, is based, as the current work, on the basic *Predictive Process Monitoring Framework* , which allows for avoiding the on-the-fly construction of the classification models by applying a clustering pre-processing phase. However, differently from the *Genetic-Enhanced Predictive Process Monitoring Framework* , the technique and hyperparameter selection is performed by manually defining the hyperparameter search space and by inspecting the returned solutions to choose the best one.

We report, in Table 2, the values of accuracy and failure-rate obtained by applying the on-the-fly approach with a manual selection of the configuration that best balances accuracy and failure-rate.[7] In Table 3, we report the validation and testing values of accuracy and failure-rate obtained using the basic *Predictive Process Monitoring Framework* with the semi-automatic selection of the configuration that best optimizes (i) accuracy, (ii) failure-rate and (iii) the balance between accuracy, failure-rate and earliness [10]. In both approaches, we used 70% of the original datasets for training and 10% for testing; in the semi-automatic approach, we also used 20% of the datasets for the tuning phase.

[7] By inspecting the results, we found the hyperparameter values that best balance accuracy and failure-rate for the on-the-fly approach. In particular, we set the confidence threshold to 0.4, the support to 4, the similarity threshold to 0.5 and the WeKa J48 confidence pruning and minimum number of instances to 0.6 and 2, respectively.

By looking at Table 2, we can observe that the accuracy values obtained with the on-the-fly approach are slightly better (up to 0.15 better) than the ones obtained on the testing set with the single-objective optimization. On the other hand, we can notice that the failure-rate values are very high (up to 0.65 higher) compared to the ones obtained with the single-objective approach. Therefore, the *Genetic-Enhanced Predictive Process Monitoring Framework* is able to get results that guarantee a better trade-off between the two quality metrics.

By looking at the results obtained with the semi-automatic approach (Table 3, balance criterion) and the single-objective approach of the *Genetic-Enhanced Predictive Process Monitoring Framework* , we can observe that they are comparable. In particular, in the case of $\phi_{11}$, using the validation set, we get a slight improvement (of 0.02) in terms of accuracy and also an improvement in terms of failure-rate (a decrease of about 0.15), while using the testing set, we obtain a worse accuracy (a decrease of 0.11) and a better failure-rate (a decrease of around 0.2). In the case of $\phi_{12}$, we observe a similar trend. The single-objective approach is able to guarantee an increase (of 0.07) in the accuracy and a decrease (of 0.09) in the failure-rate for the validation set, as well as a slight decrease (of 0.03) of both accuracy and failure-rate for the testing set. No differences in terms of results can be observed for $\phi_{21}$. In the case of $\phi_{22}$, while the accuracy remains the same for both validation and testing, the failure-rate decreases of about 0.03 for the validation set and of about 0.06 for the testing set. To summarize, the single-objective approach is able to provide solutions that are better or non-worse than the ones obtained with the semi-automatic approach.

Given these results, we can conclude that the single-objective optimization of the *Genetic-Enhanced Predictive Process Monitoring Framework* performs consistently well on the datasets under consideration, presenting testing results with an accuracy ranging between 0.73 and 0.95 and a failure-rate never higher than 0.07. Moreover, the framework is able to provide solutions that are better or non-worse than state-of-the-art approaches. A crucial aspect of this analysis is that while state-of-the-art approaches require the expertise of users in the tuning phase, by using the *Genetic-Enhanced Predictive Process Monitoring Framework* , the hyperparameter optimization does not require any human intervention. This allows us to positively answer **RQ1.2**.

In order to answer the third research question (**RQ1.3**), we evaluated the computation time required by the genetic algorithm to return the best

| Rule | Number of generations | Computation time (h) |
|---|---|---|
| $\phi_{11}$ | 20 | 38.86 |
| $\phi_{12}$ | 28 | 33.35 |
| $\phi_{21}$ | 11 | 13.35 |
| $\phi_{22}$ | 21 | 29.74 |

Table 4: Computation time (reported in hours) for each business goal required by the single-objective approach

configuration. Table 4 shows the computation time, expressed in hours, required by the algorithm for each run. Looking at Table 4, we can see how giving a direct answer to this research question is not straightforward. This is because the computation times have a significant variance, depending on the time required by the genetic algorithm to find a solution: taking into account that the terminating condition of the approach has a stagnation of 10 epochs, the time required by the algorithm varied a lot for the different business goals. For example, for the business goal $\phi_{21}$, the solution was found in the first generation of individuals, so that the time spent was only the time required to generate the successive 10 generations (needed to have the stagnation condition met). With $\phi_{22}$, the steady increase of the fitness values continued to reset the termination condition, requiring ten additional generations.

Comparing $\phi_{11}$ and $\phi_{22}$, which have almost the same number of generations, we can notice that the time spent to find a solution using the first dataset is higher than the one spent for the second dataset. A possible cause for such a difference can be found in the different sizes of the two datasets, as also observed in [10]: indeed, the first dataset contains almost three times the number of events contained in the second one.

Table 5 shows the details of the run using the business goal $\phi_{12}$, with the time spent for each generation of the algorithm. In particular, we can observe that the initial generations have computation times substantially higher than the forthcoming ones, and that, in this case, after the fifth generation the time spent settles on around 0.59 hours (about 35 minutes) per generation, which is a reasonable time for a tuning algorithm.

Note that, when dealing with datasets that are larger than the ones we used in this paper, training and validation for each configuration, can be distributed. The execution of each configuration, indeed, is actually independent of the execution of the other configurations of the same generation, so that Big Data technologies like Hadoop, Spark and Flink can be used.

| Generation | Computation time (h) | Generation | Computation time (h) |
|---|---|---|---|
| 0 | 7.37 | 14 | 0.54 |
| 1 | 8.12 | 15 | 0.54 |
| 2 | 1.27 | 16 | 0.55 |
| 3 | 0.88 | 17 | 0.58 |
| 4 | 1.03 | 18 | 0.59 |
| 5 | 1.75 | 19 | 0.61 |
| 6 | 0.59 | 20 | 0.62 |
| 7 | 0.58 | 21 | 0.63 |
| 8 | 0.56 | 22 | 0.63 |
| 9 | 0.57 | 23 | 0.63 |
| 10 | 0.54 | 24 | 0.63 |
| 11 | 0.55 | 25 | 0.63 |
| 12 | 0.54 | 26 | 0.63 |
| 13 | 0.53 | 27 | 0.63 |

Table 5: Computation time required by the single-objective approach for each generation for the business goal $\phi_{12}$

| Rule | Training time (s) | Avg. prediction time (ms) |
|---|---|---|
| $\phi_{11}$ | 18.09 | 0.48 |
| $\phi_{12}$ | 61.89 | 0.54 |
| $\phi_{21}$ | 57.74 | 8.75 |
| $\phi_{22}$ | 365.7 | 230.63 |

Table 6: Training and testing time required for each business goal (configurations selected using the single-objective approach)

These observations allow us to positively answer **RQ1.3**. Overall, we can conclude that the single-objective approach is able to identify in a reasonable amount of time a (near-optimal) configuration that is able to provide accurate predictions (**RQ1**).

Once the near-optimal configuration for the specific dataset and business goal has been identified by the algorithm, it can be used for training the system and making predictions for new unseen traces. Table 6 reports the total time required by the system for the training phase, as well as the average time required for a single prediction. The results show that training can be performed in the order of minutes, while online prediction demands less than one second per trace. Once identified the best configuration for a new dataset, both the training and the actual online predictions can therefore be performed relatively fast.

### *7.4.2. Multi-Objective Approach Experimental Results*

In order to answer **RQ.2.1**, the plots of the Pareto front returned by the multi-objective approach for each business goal are reported. Figures 9 and 10 show the plot of the Pareto front of the four different runs (with $\phi_{11}$, $\phi_{12}$, $\phi_{21}$, and $\phi_{22}$).

By analyzing the shape of the four plots, we can immediately see a significant difference between the plots related to $\phi_{11}$ and $\phi_{12}$ and the plots related to $\phi_{21}$ and $\phi_{22}$. First, the number of configurations returned for the first dataset is much higher than the one returned for the second dataset. Second, the points on the Pareto related to the first dataset are more widely distributed than the ones related to the second dataset. These two considerations show that the first dataset provides to the user a wider choice than the second dataset on the configurations that best suit her needs. In addition, from the plot related to $\phi_{12}$, it is evident that, in this case, the set of solutions returned by the multi-objective algorithm, which is obtained with a higher number of generations, results in a more defined shape of the Pareto front. These observations answer **RQ2.1**.

As with the single-objective approach, in order to answer **RQ2.2**, we evaluated the Pareto of configuration solutions obtained with the multi-objective approach on a set of unseen traces (testing set). Starting from the Pareto of solutions, we selected three different types of configurations: a solution with high accuracy, a solution with low failure-rate and a solution balancing the two metrics.[8]

The first observation that can be done is that the Pareto shape influences the choice of the best configuration for a user. In the first two plots, indeed, the user would have no difficulties in finding a configuration which satisfies her needs, whatever they are (high accuracy, low failure-rate or a trade-off between the two). In the latter two plots, instead, the low number of points concentrated in the same area constrains the user's choice. This results in solutions that, although selected to optimize different objective functions (i.e., maximizing accuracy, minimizing failure-rate or balancing), are very similar to each other. This flattening of the solution space already appeared in the case of the single-objective approach, which was able to identify the

[8] In particular, from the Pareto front, we removed the outliers and we selected the point with the highest accuracy, the point with the lowest failure-rate and the one that balances accuracy and failure-rate.

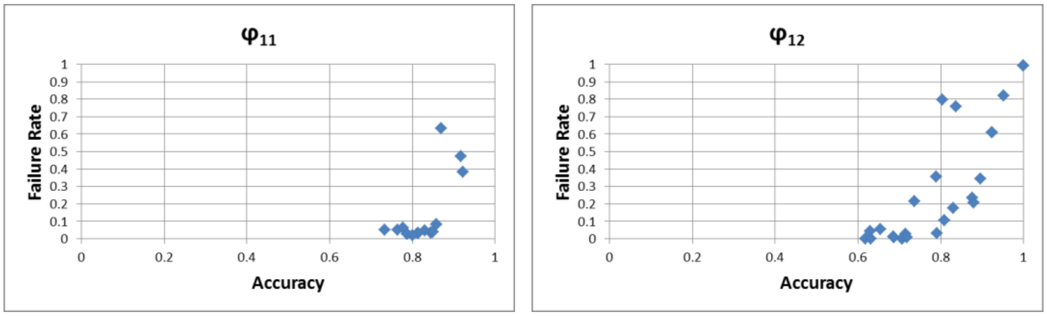


Figure 9: Pareto fronts obtained for the BPI Challenge 2011 dataset

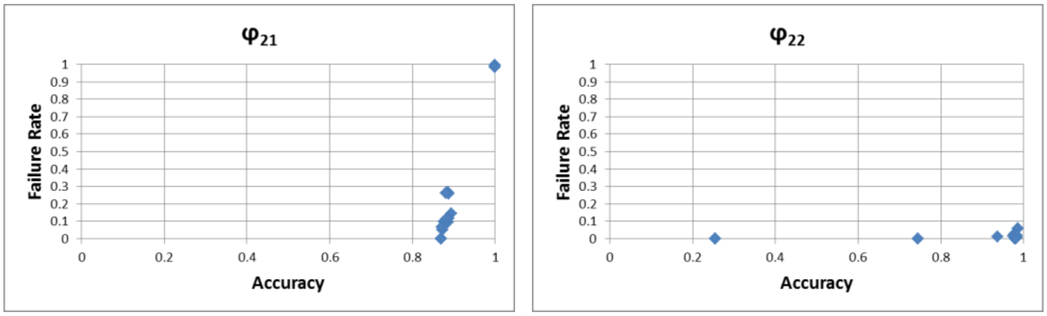


Figure 10: Pareto fronts obtained for the BPI Challenge 2015 dataset

best fitness value already at the first generation.

By looking at the data reported in Table 7, we can analyze the validation and testing results obtained using the multi-objective approach. In particular, by inspecting the data related to the first two business goals, we can notice that the results are overall good, with scores for each configuration metric that allow the user to obtain good results, according to the selected objective. For example, if we want to target a high accuracy, the solutions have an accuracy value of around 0.92 on the validation set that, on the testing set, decreases for the first business goal (with a difference of about 0.22) and increases for the second one (the difference is of around 0.02). A similar trend can also be observed for the failure-rate: for the first business goal, we have an increase, while for the second one a marginal decrease. Looking at the solutions focusing on the best failure-rate, we can observe a slight variation of the accuracy and of the failure-rate from the validation to the testing set (in the order of few hundredths). Only a slight difference exists between the validation set and the testing set also when considering as target

| Rule | Objective | Metrics | Validation Results | Testing Results | Metrics Difference |
|---|---|---|---|---|---|
| $\phi_{11}$ | Accuracy | Accuracy | 0.9219 | 0.7069 | -0.2151 |
| | | Failure-Rate | 0.3816 | 0.4912 | +0.1096 |
| | Failure-Rate | Accuracy | 0.8009 | 0.7632 | -0.0377 |
| | | Failure-Rate | 0.0088 | 0 | -0.0088 |
| | Balance | Accuracy | 0.8578 | 0.8286 | -0.0292 |
| | | Failure-Rate | 0.0075 | 0.0789 | +0.0044 |
| $\phi_{12}$ | Accuracy | Accuracy | 0.9231 | 0.94 | +0.0169 |
| | | Failure-Rate | 0.6009 | 0.5614 | -0.0395 |
| | Failure-Rate | Accuracy | 0.7181 | 0.7387 | +0.0206 |
| | | Failure-Rate | 0.0044 | 0.0263 | +0.0219 |
| | Balance | Accuracy | 0.8798 | 0.8539 | -0.0259 |
| | | Failure-Rate | 0.1974 | 0.2193 | +0.0219 |
| $\phi_{21}$ | Accuracy | Accuracy | 0.8947 | 0.9068 | -0.012 |
| | | Failure-Rate | 0.1255 | 0.0248 | -0.1007 |
| | Failure-Rate | Accuracy | 0.8703 | 0.9091 | +0.0388 |
| | | Failure-Rate | 0 | 0 | 0 |
| | Balance | Accuracy | 0.8947 | 0.9068 | +0.0121 |
| | | Failure-Rate | 0.1255 | 0.0248 | -0.1007 |
| $\phi_{22}$ | Accuracy | Accuracy | 0.9867 | 1 | +0.0133 |
| | | Failure-Rate | 0.0586 | 0.1653 | +0.1067 |
| | Failure-Rate | Accuracy | 0.9791 | 0.9669 | -0.0121 |
| | | Failure-Rate | 0 | 0 | 0 |
| | Balance | Accuracy | 0.9831 | 0.9339 | -0.0492 |
| | | Failure-Rate | 0.0084 | 0 | -0.0084 |

Table 7: Results obtained with the validation and testing sets using the multi-objective approach

the balance of the two metrics. Overall, except for the accuracy value of the first business goal when optimizing on the accuracy, only minor variations can be observed between validation and testing results. Therefore, we can assess that, for the first dataset, the results obtained in the testing phase are quite close to the ones obtained in the tuning phase.

If we analyze the other two business goals, we can observe that, for both validation and testing sets, the accuracy values range between 0.87 and 1, while the failure-rate scores are very close to zero. By inspecting the solutions that look at a high accuracy, we can notice some fluctuations between the validation and the testing results in terms of failure-rate (increasing up to 0.11 for $\phi_{22}$) from the tuning to the testing phase. When considering as target a low failure-rate, the results are satisfactory both in terms of accuracy and of failure-rate. Finally, when focusing on configurations able to balance the two metrics, the testing results show either improvements with respect to the validation results or only slight variations. Also in this second dataset, therefore, except for few cases, validation and testing results do not present significant differences.

We compared the results obtained with the multi-objective approach of the *Genetic-Enhanced Predictive Process Monitoring Framework* with the ones obtained with the on-the-fly approach [13] and with the semi-automatic approach [10]. Again, we used accuracy and failure-rate for evaluating the quality of the predictions.

In particular, we compared the results of the on-the-fly approach, whose hyperparameters have been manually tuned (Table 2), with the ones obtained with the multi-objective approach by balancing the accuracy and the failure-rate in the testing phase. We can observe that the multi-objective approach is able to produce solutions that either dominate or improve in one of the two objectives the solutions of the on-the-fly approach. However, while the improvement in terms of failure-rate decrease is on average in the order of few tenths, the decrease in terms of accuracy is always in the order of hundredths. As for the single-objective approach, therefore, we can assess that also the solutions obtained with the multi-objective approach are non-worse than the solutions obtained with the on-the-fly approach.

We also compared the results obtained with the multi-objective approach of the *Genetic-Enhanced Predictive Process Monitoring Framework* with the ones obtained with the semi-automatic approach [10]. In particular, we compared accuracy and failure-rate values of the solution provided by the multi-objective approach maximizing the accuracy (resp. failure-rate and balancing the two objectives) with the ones of the solution obtained with the basic *Predictive Process Monitoring Framework* by manually inspecting different configurations and selecting the one that maximizes accuracy (resp. failure-rate and balancing the two objectives). By looking at the differences between the results in Table 3 and those in Table 7, we can notice that, except for one case, the multi-objective approach always improves the validation results at least in one of the objectives. In the validation set, indeed, we find that the accuracy increases, or the failure-rate decreases, or both the objectives are improved. The only exception is the solution optimizing the failure-rate for $\phi_{12}$, although the difference between the two values is marginal (a decrease of 0.03 for the accuracy and an increase of 0.004 for the failure-rate). We can observe that, also in the testing results, for more than half of the settings, at least one of the two objectives is improved by using the multi-objective approach. For all the remaining cases, the difference between the values of

| Rule | Computation time (h) |
|---|---|
| $\phi_{11}$ | 51.32 |
| $\phi_{12}$ | 65.28 |
| $\phi_{21}$ | 42.42 |
| $\phi_{22}$ | 18.49 |

Table 8: Computation time (reported in hours) required by the multi-objective approach for each business goal

| Generation | Computation time (h) | Generation | Computation time (h) |
|---|---|---|---|
| 0 | 6.36 | 8 | 2.81 |
| 1 | 1.93 | 9 | 4.07 |
| 2 | 2.92 | 10 | 2.26 |
| 3 | 1.99 | 11 | 3.69 |
| 4 | 1.67 | 12 | 2.70 |
| 5 | 2.44 | 13 | 2.18 |
| 6 | 2.63 | 14 | 2.26 |
| 7 | 2.49 | | |

Table 9: Computation time required by the multi-objective approach for each generation for the business goal $\phi_{21}$

accuracy and failure-rate is marginal (always lower than 0.18). Therefore, we can assess that also in the multi-objective case, the obtained results are quite close to the ones obtained with the semi-automatic approach.

To sum up, the results returned by the multi-objective approach of the *Genetic-Enhanced Predictive Process Monitoring Framework* are in line with the selected criteria and do not reveal significant differences between the validation and the testing results, thus indicating that the solutions found are reliable and could be used in new unseen cases with good confidence. Furthermore, the obtained results are comparable with the results obtained with state-of-the-art approaches, which, however, demands for human intervention and expertise in the tuning phase. These observations allow us to positively answer **RQ2.2**.

Answering **RQ2.3** requires an analysis of the computation time for each business goal shown in Table 8. Comparing these results with the ones obtained with the single-objective approach, we can notice how, except for $\phi_{22}$, all the computation times are notably higher in the multi-objective case.

In order to give a better picture of the internal computation times, Table 9 shows the time spent for each generation of the run involving the business goal $\phi_{21}$. We observe that, as in the single-objective approach, the time spent

| Rule | Objective | Training time (s) | Avg. prediction time (ms) |
|---|---|---|---|
| $\phi_{11}$ | Accuracy | 28.63 | 12.54 |
| | Failure-Rate | 30.07 | 1.23 |
| | Balance | 19.53 | 2.75 |
| $\phi_{12}$ | Accuracy | 348.91 | 1915.14 |
| | Failure-Rate | 56.09 | 1.36 |
| | Balance | 30.35 | 3.21 |
| $\phi_{21}$ | Accuracy | 108.24 | 272.74 |
| | Failure-Rate | 1024.76 | 24.67 |
| | Balance | 222.9 | 25.98 |
| $\phi_{22}$ | Accuracy | 117.84 | 110.74 |
| | Failure-Rate | 1058.73 | 25.6 |
| | Balance | 768.45 | 0.25 |

Table 10: Training and testing time required for each business goal (configurations selected using the multi-objective approach)

for the first generation is higher than the ones spent for all the others. This may be due to the initial setting of the algorithm that requires some time, or to the framework initialization. For all the other generations, we have a computation time of about 2.7 hours (~2 hours an 45 minutes), though with some variations across the generations.

Also in the case of the multi-objective approach, the scalability can be improved for possibly larger datasets by exploiting Big data technologies like Apache Hadoop, Spark or Flink. Indeed, training and validation of the different configurations, which are the most time consuming parts of the approach, can be distributed.

We can therefore conclude that, although more expensive than the single-objective approach in terms of time spent, the multi-objective approach requires an average time per generation that is reasonable for a task such as the hyperparameter selection, thus allowing us to answer **RQ2.3** positively. Overall, we can conclude that the multi-objective approach is able to provide users with a set of near-optimal but suitable framework configurations, from which they can select the one(s) that best suit their requirements. The execution times of the approach are in the order of a few hours, which is suitable for overnight processing. Given that the selection of the instance framework does not need to be performed frequently and may even be a one-off operation for a given dataset, this performance is within manageable ranges (**RQ2**).

Also in the case of the multi-objective approach, once one or more configurations for the specific dataset and business goal have been selected, they can be used for training the system and making predictions for new unseen traces. Table 10 reports the total time required by the system for the training phase and the average time required for a single prediction for each of the three types of configuration (the one with high accuracy, the one with low failure-rate and the one balancing the two metrics). The results show that training can be performed in at most 20 minutes, while online prediction demands for less than two seconds per trace. The difference in terms of performance can mainly be ascribed to the clustering algorithm selected. When dbscan is used, the prediction time is higher (in the case of accuracy optimization for $\phi_{12}$, in all the optimizations for $\phi_{21}$ and in the cases of accuracy optimization and failure-rate optimization for $\phi_{22}$). Therefore, once identified the best configuration for a new dataset, both the training and the actual online predictions can be performed in an amount of time that is still reasonable.

### *7.5. Summary and discussion*

To sum up, both the investigated approaches provide high quality results without requiring human intervention in the selection of the most suitable techniques and in the hyperparameter tuning for a given dataset. Moreover, the time required by the two proposed approaches for the selection of the most suitable configuration is still reasonable for real-life datasets. By comparing the two proposed approaches, it is clear that the multi-objective approach requires higher execution times than the single-objective approach though this was expected since the multi-objective approach provides a richer output.

### *7.6. Threats to Validity*

Despite the effort spent to generate a set of results that are not biased by external factors that would reduce their validity, some aspects of this work are prone to this kind of problems. Three main threats affect the validity of the presented results: (i) potential overfitting, (ii) limited number of datasets and business goals used, and (iii) fixed number of generations in the multi-objective approach.

The lack of a cross-validation testing during the tuning phase could cause a potential overfitting of the solutions, thus potentially implying a reduction of the performance of the approaches on unseen data. Nevertheless, the narrow differences existing between validation and testing results in most of the cases and for both approaches, suggest that the models, in general, do not overfit the validation sets.

Concerning the second threat, the approach would have benefit of a wider experimentation. Indeed, it has been applied to two datasets and to two business goals only for each dataset. This threat is mitigated by the fact that the two datasets are real-life ones and the investigated business goals are realistic problems that a user could be interested in.

Finally, in the case of the multi-objective approach, the number of generations is fixed: this could generate Pareto fronts that are incomplete or that have not yet reached the convergence (as happened in the first business goal of Figure 7), thus returning a set of configurations that could be not yet the best one. Nevertheless, the results obtained with such a fixed number of generations are still satisfactory.

## 8. Related Work

Two bodies of previous work are related to this paper: those concerning predictive business process monitoring and those related to hyperparameter optimization.

As for the first branch, we can classify the existing works based on the techniques used for making predictions. Some of the works existing in the literature rely on lazy approaches like Case-Based Reasoning (CBR), i.e., they use past data to make predictions in a lazy way; other works rely on eager approaches.

In the former group, we mainly find applications of CBR in the healthcare domain [39, 40] and in the industrial domain [41] (e.g., in the context of predictive maintenance). In the second group, instead, we find several works related to predictions in the process monitoring field. Eager approaches can be further classified based on the type of generated predictions.

A first category deals with approaches for time-based predictions. In [14], the authors present a family of approaches in which annotated transition systems, containing time information extracted from event logs, are used to: (i) check time conformance, (ii) predict the remaining processing time, and (iii) recommend appropriate activities to end users. In [15], an ad-hoc clustering approach for predicting process performance measures is presented, in which context-related execution scenarios are discovered and modeled through state-aware performance predictors. In [16], stochastic Petri nets are used to predict the remaining execution time of a process. In [42], the authors present a method for predicting the remaining processing time that relies on a bi-dimensional feature space. The first dimension expresses intra-case dependencies, while the second dimension represents inter-case dependencies capturing the interplay of all running cases.

Other works in the predictive business process monitoring literature focus on approaches that generate predictions and recommendations to reduce risks. For example, in [17], the authors present a technique to support process participants in making risk-informed decisions with the aim of reducing process failures. In [18], the authors make predictions about time-related process risks by identifying and exploiting statistical indicators that highlight the possibility of transgressing deadlines. In [19], an approach for Root Cause Analysis through classification algorithms is presented.

A third group of approaches in the process monitoring field predicts the outcome (e.g., the satisfaction of a business objective) of a process. In [13] an approach is introduced, which is able to predict the fulfillment (or the violation) of a boolean predicate in a running case, by looking at: (i) the sequence of activities already performed in the case; and (ii) the data payload of the last activity of the running case. The approach, which provides accurate results at the expense of a high runtime overhead, has been enhanced in [1] by introducing a clustering preprocessing step in which cases sharing a similar activity history are clustered together. A classifier for each cluster is trained with the data payload of the traces in the cluster. In [23], the authors compare different feature encoding approaches where traces are treated as complex symbolic sequences, i.e., sequences of activities each carrying a data payload consisting of attribute-value pairs. In [43], the approach in [23] has been extended by clustering the historical traces before classification. In [44], unstructured information contained in text messages exchanged during process executions has been leveraged for improving the prediction accuracy. In [20], a comparison of the existing outcome-based predictive monitoring approaches is presented.

A key difference between these approaches and the approach presented in this paper is that we provide a general, customizable framework for predictive business process monitoring that is flexible and can be implemented in different variants with different sets of configurable techniques. Given the high number of possible resulting configurations, the *Genetic-Enhanced Predictive Process Monitoring Framework* also supports users in the tuning phase.

Recently, a few studies have investigated the use of deep learning techniques – specifically techniques based on LSTM neural networks – to tackle predictive process monitoring problems such as predicting the next event in a trace, and the timestamp of the next event, or predicting the remaining time until completion of an ongoing case [45, 46, 47]. These studies have shown that when the datasets are large, deep learning techniques can outperform techniques based on classical machine learning techniques. Since the aim of this paper is to provide a framework that auto-tunes itself to different datasets (including smaller datasets), we do not consider the possibility of using deep learning techniques. However, the framework is designed in such a way that it can be used in conjunction with deep learning classifiers.

Regarding the works related to hyperparameter optimization, several approaches in machine learning have been proposed for the selection of a learning algorithm [48], for the tuning of hyperparameters [49], and for the combined optimization of both the algorithm and the hyperparameters [5]. The problem we address in this paper is to tune both the machine learning algorithm and the hyperparameter values. One of the first works falling in the same category is Auto-WEKA [5]. The idea of this work is to map the problem of algorithm selection to that of hyperparameter optimization and to approach the latter problem based on a sequential model-based optimization and a random forest regression model. MLbase [50] also addresses the same problem as Auto-WEKA and approaches it using distributed data mining algorithms.

Differently from these approaches, the problem that we face in this work is more complex. Indeed, in the predictive monitoring of business processes, we have to deal with predictions over complex symbolic sequences. Therefore, we have more than one sub-problem (e.g., clustering and classification) and these sub-problems depend on each other. Therefore, the algorithm (and hyperparameters) optimization for a sub-problem cannot be defined independently of the other. In this scenario, genetic algorithms represent a solution that guarantee a good the quality of the results and reasonable execution times. Other works in the literature use evolutionary algorithms in the context of classification problems. For instance, in [7, 8], genetic algorithms have been used for the automatic tuning of SVM hyperparameters. In [9], a genetic algorithm has been used for the hyperparameter optimization of neural networks. In [51], a hyper-heuristic evolutionary algorithm has been proposed for evolving design components of top-down decision tree induction algorithms according to the specific classification dataset considered. A hybrid learning methodology that integrates genetic algorithms and decision tree learning is introduced in [52], where optimal sub-sets of discriminatory features are evolved for robust pattern classification.

## 9. Conclusion

We have presented a framework for predictive business process monitoring armed with two hyperparameter optimization techniques. The proposed techniques rely on genetic algorithms to scalably search through the hyperparameter space in order to identify configurations that either maximize a single-objective function, or compute Pareto fronts from which the user can select a given configuration depending on the desired tradeoffs. In particular, we have shown how genetic algorithms can be successfully used to solve two combined sub-problems: clustering (needed to extract information from the control flow perspective of business process executions) and classification (to get information from the data payloads attached to each event). Then, we have demonstrated that the customized genetic algorithms allow us to (i) reduce the execution times for hyperparameter optimization, and (ii) find a balance in the optimization of the different dimensions that are involved in the predictive monitoring task (e.g., accuracy, earliness, and failure-rate).

We have documented and supported the suitability of the proposed framework through an extensive experimentation by comparing it with all existing alternatives. The results confirm the scalability of the approach and its ability to identify accurate and reliable framework configurations.

In the experimentation, we have focused on predicting binary outcomes (e.g., a case can be positive or negative). An avenue for future work is to design similar frameworks for other types of predictive process monitoring tasks, such as predicting the remaining time or the cost of a case, predicting the next activity(ies) in a case, or estimating the probability that a given activity will be executed as a part of an ongoing case [53]. Moreover, it would be interesting to apply the same approach for hyperparameter optimization in the cases of multi-class (i.e., when traces can be classified using more than two classes) and multi-labeling (i.e., when the same trace can be associated to more than one label) classification, so as to investigate how the framework behaves when using these types of algorithm (e.g., whether the convergence of the genetic algorithm is slower than in the case of binary classification). Furthermore, other evolutionary techniques can be explored in order to evaluate their advantages and drawbacks with respect to the use of genetic algorithms. Finally, another avenue for future work is that of instantiating the proposed framework using other machine learning techniques. For example, deep learning approaches can be used when dealing with large datasets or when temporal features have to be taken into consideration since they are proven to be more efficient for handling complex temporal relations [54, 55].

**Acknowledgments**. This research is partly supported by the Estonian Research Council (grant IUT20-55).

## References

[1] C. Di Francescomarino, M. Dumas, F. M. Maggi, I. Teinemaa, Clustering-based predictive process monitoring, IEEE Transactions on Services Computing PP (99) (2017) 1–1. doi:10.1109/TSC.2016.2645153.

[2] M. Federici, W. Rizzi, C. D. Francescomarino, M. Dumas, C. Ghidini, F. M. Maggi, I. Teinemaa, A ProM operational support provider for predictive monitoring of business processes, in: Proceedings of the BPM Demo Session 2015, CEUR-WS.org, 2015, pp. 1–5.

[3] J. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, Algorithms for hyper-parameter optimization, in: Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain., 2011, pp. 2546–2554.

[4] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, Journal of Machine Learning Research 13 (2012) 281–305.

[5] C. Thornton, F. Hutter, H. H. Hoos, K. Leyton-Brown, Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms, in: Proc. of KDD-2013, 2013, pp. 847–855.

[6] F. Hutter, J. Lücke, L. Schmidt-Thieme, Beyond manual tuning of hyperparameters, KI - Künstliche Intelligenz 29 (4) (2015) 329–337. doi:10.1007/s13218-015-0381-0.

[7] S. Lessmann, R. Stahlbock, S. Crone, Optimizing hyperparameters of support vector machines by genetic algorithms, Vol. 1, CSREA Press, 2005, pp. 74–82.

[8] S. Di Martino, F. Ferrucci, C. Gravino, F. Sarro, A Genetic Algorithm to Configure Support Vector Machines for Predicting Fault-Prone Components, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 247–261. doi:10.1007/978-3-642-21843-9_20.

[9] S. R. Young, D. C. Rose, T. P. Karnowski, S.-H. Lim, R. M. Patton, Optimizing deep learning hyper-parameters through an evolutionary algorithm, in: Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments, MLHPC '15, ACM, New York, NY, USA, 2015, pp. 4:1–4:5. doi:10.1145/2834892.2834896.

[10] C. Di Francescomarino, M. Dumas, M. Federici, C. Ghidini, F. M. Maggi, W. Rizzi, Predictive business process monitoring framework with hyperparameter optimization, in: Proc. of CAiSE, 2016, pp. 361–376.

[11] C. Nyce, Predictive analytics white paper, in: American Institute for Chartered Property Casualty Underwriters/Insurance Institute of America, 2007, p. 1.

[12] W. Eckerson, Extending the value of your data warehousing investment, in: The Data Warehouse Institute, 2007, p. 1.

[13] F. M. Maggi, C. Di Francescomarino, M. Dumas, C. Ghidini, Predictive monitoring of business processes, in: Proc. of CAiSE 2014, Vol. 8484 of LNCS, Springer, 2014, pp. 457–472.

[14] W. M. P. van der Aalst, M. H. Schonenberg, M. Song, Time prediction based on process mining, Information Systems 36 (2) (2011) 450–475.

[15] F. Folino, M. Guarascio, L. Pontieri, Discovering context-aware models for predicting business process performances, in: Proc. of On the Move to Meaningful Internet Systems (OTM), Springer, 2012, pp. 287–304.

[16] A. Rogge-Solti, M. Weske, Prediction of remaining service execution time using stochastic petri nets with arbitrary firing delays, in: Proc. of ICSOC, Springer, 2013, pp. 389–403.

[17] R. Conforti, M. de Leoni, M. L. Rosa, W. M. P. van der Aalst, Supporting risk-informed decisions during business process execution, in: Proc. of CAiSE 2013, Springer, 2013, pp. 116–132.

[18] A. Pika, W. Aalst, C. Fidge, A. Hofstede, M. Wynn, Predicting deadline transgressions using event logs, in: Proc. of the BPM'2012 Workshops, Springer, 2013, pp. 211–216.

[19] S. Suriadi, C. Ouyang, W. Aalst, A. Hofstede, Root cause analysis with enriched process logs, in: Proc. of BPM Workshops, Vol. 132 of LNBIP, Springer, 2013, pp. 174–186.

[20] I. Teinemaa, M. Dumas, M. L. Rosa, F. M. Maggi, Outcome-oriented predictive process monitoring: Review and benchmark, CoRR abs/1707.06766.

[21] M. Mitchell, Genetic algorithms: An overview, Complexity 1 (1) (1995) 31–39.

[22] F. P. Preparata, M. I. Shamos, Computational Geometry - An Introduction, Texts and Monographs in Computer Science, Springer, 1985.

[23] A. Leontjeva, R. Conforti, C. Di Francescomarino, M. Dumas, F. M. Maggi, Complex symbolic sequence encodings for predictive monitoring of business processes, in: Proc. of BPM, Springer, pp. 297–313.

[24] D. A. Grossman, O. Frieder, Information Retrieval: Algorithms and Heuristics, Springer, 2004.
URL http://www.springer.com/computer/book/978-1-4020-3003-1

[25] P. Fournier-Viger, J. C.-W. Lin, R. U. Kiran, Y. S. Koh, Data Science and Pattern Recognition.

[26] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, in: Proc. of KDD, 1996, pp. 226–231.

[27] J. MacQueen, Some methods for classification and analysis of multivariate observations, in: Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics, University of California Press, Berkeley, Calif., 1967, pp. 281–297.

[28] L. Rokach, O. Maimon, Clustering Methods, Springer US, Boston, MA, 2005, pp. 321–352. doi:10.1007/0-387-25465-X_15.

[29] A. Strehl, J. Ghosh, Cluster ensembles—a knowledge reuse framework for combining multiple partitions, Journal of machine learning research 3 (Dec) (2002) 583–617.

[30] M. Westergaard, F. Maggi, Modelling and Verification of a Protocol for Operational Support using Coloured Petri Nets, in: Proc. of the International Conference on Applications and Theory of Petri Nets (ATPN), 2011.

[31] F. Maggi, M. Westergaard, Designing software for operational decision support through coloured petri nets, Enterprise Information Systems 0 (0) (0) 1–21.

[32] F. Salfner, M. Lenk, M. Malek, A survey of online failure prediction methods, ACM Comput. Surv. 42 (3) (2010) 10:1–10:42.

[33] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: Nsga-ii, Trans. Evol. Comp 6 (2) (2002) 182–197.

[34] C. Chatelain, S. Adam, Y. Lecourtier, L. Heutte, T. Paquet, Multiobjective optimization for svm model selection, in: Ninth International Conference on Document Analysis and Recognition (ICDAR 2007), Vol. 1, IEEE, 2007, pp. 427–431.

[35] E. Zitzler, M. Laumanns, L. Thiele, SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization, in: K. C. Giannakoglou, D. T. Tsahalis, J. Périaux, K. D. Papailiou, T. Fogarty (Eds.), Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems, International Center for Numerical Methods in Engineering, Athens, Greece, 2001, pp. 95–100.

[36] 3TU Data Center, BPI Challenge 2011 Event Log (2011). doi:doi:10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54.

[37] B. van Dongen;, Bpi challenge 2015 (2015). doi:10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1.

[38] A. Pnueli, The temporal logic of programs, in: 18th Annual Symposium on Foundations of Computer Science, 1977, pp. 46–57.

[39] R. Janssen, P. Spronck, A. Arntz, Case-based reasoning for predicting the success of therapy, Expert Systems 32 (2) (2015) 165–177, eXSY-Mar-12-057.R2. doi:10.1111/exsy.12074.

[40] L. Sukumar, M. Zulkefli, N. L. W. Yan, L. K. Meng, N. F. I. Tahir, Predictive analytic in health care using case-based reasoning (CBR), Zulikha, J. and N. H. Zakaria (Eds.), 2017, pp. 8–15.

[41] N. Xiong, P. Funk, T. Olsson, Case-Based Reasoning Supports Fault Diagnosis Using Sensor Information, 2012.

[42] A. Senderovich, C. Di Francescomarino, C. Ghidini, K. Jorbina, F. M. Maggi, Intra and inter-case features in predictive process monitoring: A tale of two dimensions, in: Business Process Management - 15th International Conference, BPM 2017, Barcelona, Spain, September 10-15, 2017, Proceedings, 2017, pp. 306–323.

[43] I. Verenich, M. Dumas, M. L. Rosa, F. M. Maggi, C. D. Francescomarino, Complex symbolic sequence clustering and multiple classifiers for predictive process monitoring, in: Business Process Management Workshops - BPM 2015, 13th International Workshops, Innsbruck, Austria, August 31 - September 3, 2015, Revised Papers, 2015, pp. 218–229.

[44] I. Teinemaa, M. Dumas, F. M. Maggi, C. Di Francescomarino, Predictive business process monitoring with structured and unstructured data, in: Business Process Management - 14th International Conference, BPM 2016, Rio de Janeiro, Brazil, September 18-22, 2016. Proceedings, 2016, pp. 401–417.

[45] N. Tax, I. Verenich, M. L. Rosa, M. Dumas, Predictive business process monitoring with LSTM neural networks, in: Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE), 2017, pp. 477–492.

[46] J. Evermann, J. Rehse, P. Fettke, Predicting process behaviour using deep learning, Vol. 100, 2017, pp. 129–140.

[47] C. Di Francescomarino, C. Ghidini, F. M. Maggi, G. Petrucci, A. Yeshchenko, An eye into the future: Leveraging a-priori knowledge

in predictive business process monitoring, in: Business Process Management - 15th International Conference, BPM 2017, Barcelona, Spain, September 10-15, 2017, Proceedings, 2017, pp. 252–268.

[48] B. Pfahringer, H. Bensusan, C. Giraud-Carrier, Meta-learning by landmarking various learning algorithms, in: In Proceedings of the Seventeenth International Conference on Machine Learning, Morgan Kaufmann, 2000, pp. 743–750.

[49] F. Hutter, H. H. Hoos, K. Leyton-Brown, Sequential model-based optimization for general algorithm configuration, in: Learning and Intelligent Optimization, Springer, 2011, pp. 507–523.

[50] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, M. I. Jordan, Mlbase: A distributed machine-learning system, in: Proc. of CIDR, 2013.

[51] R. C. Barros, M. P. Basgalupp, A. A. Freitas, A. C. P. L. F. de Carvalho, Evolutionary design of decision-tree algorithms tailored to microarray gene expression data sets, IEEE Transactions on Evolutionary Computation 18 (6) (2014) 873–892. doi:10.1109/TEVC.2013.2291813.

[52] J. W. Bala, J. Huang, H. Vafaie, K. DeJong, H. Wechsler, Hybrid learning using genetic algorithms and decision trees for pattern classification, in: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes, 1995, pp. 719–724.

[53] I. Verenich, M. Dumas, M. La Rosa, F. M. Maggi, C. Di Francescomarinos, Minimizing overprocessing waste in business processes via predictive activity ordering, in: Proceedings of CAiSE, 2016, pp. 186–202.

[54] G. W. Taylor, R. Fergus, Y. LeCun, C. Bregler, Convolutional Learning of Spatio-temporal Features, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 140–153.

[55] F. J. O. Morales, D. Roggen, Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition, in: Sensors, 2016.