Masterarbeit

HYPEX: Explainable Hyperparameter Optimization in Time Series Anomaly Detection

HYPEX: Interpretierbare Parameter Optimierung für Anomalie Erkennung auf Zeitserien

by
Mats Pörschke

Supervisors
Prof. Dr. Felix Naumann
Information Systems Chair
Prof. Dr. Thorsten Papenbrock
Philipps-Universität Marburg

Hasso Plattner Institute at University of Potsdam

September 14, 2022

Abstract

Adapting a complex system's configuration parameters to a specific application is a particularly difficult task. It requires a deep understanding of the system's internal structures to know the impact of configuration parameter changes on inner workings. Existing iterative approaches automate the time-intensive parameter optimization. However, they require the user to provide labelled datasets to benchmark suggested configuration parameters before applying them onto real-world application data. In many domains, labelling large, complex datasets by human experts is not feasible. We therefore propose a fully automated hyperparameter optimization framework that identifies promising hyperparameters on synthetically generated datasets and learns parameter dependencies that enable easy parameter adjustments to new, unlabeled and yet unseen datasets. We demonstrate the capabilities of our work in the context of time series anomaly detection where our parameter suggestions made on unseen data achieve significant performance increases compared to existing manual approaches.

Zusammenfassung

Die Anpassung der Konfigurationsparameter eines komplexen Systems an eine bestimmte Anwendung ist eine besonders schwierige Aufgabe. Sie erfordert ein tiefes Verständnis der internen Strukturen des Systems, um die Auswirkungen von Änderungen der Konfigurationsparameter auf die inneren Abläufe zu kennen. Bestehende iterative Ansätze automatisieren die zeitintensive Parameteroptimierung, erfordern jedoch, dass der Benutzer gelabelte Datensätze bereitstellt, um die vorgeschlagenen Konfigurationsparameter zu testen, bevor sie auf reale Anwendungsdaten angewendet werden. In vielen Bereichen ist das Erstellen von Annotationen für große, komplexe Datensätze durch menschliche Experten nicht machbar. Wir schlagen daher ein vollautomatisches Verfahren zur Optimierung von Hyperparametern vor, das vielversprechende Hyperparameter auf synthetisch erzeugten Datensätzen identifiziert und Parameterabhängigkeiten erlernt, die eine einfache Anpassung der Parameter an neue, nicht gekennzeichnete und noch nicht gesehene Datensätze ermöglichen. Wir demonstrieren die Fähigkeiten unserer Arbeit im Zusammenhang mit der Erkennung von Zeitreihenanomalien, bei der unsere Parametervorschläge auf ungesehenen Daten eine signifikante Leistungssteigerung im Vergleich zu bestehenden manuellen Ansätzen erzielen.

Contents

1 The Curse of Hyperparameters

Tuning a system or algorithm to achieve decent run time performance or accuracy metrics involves adapting existing configuration parameters to a specific application scenario. We find this to be a common task known from various application domains where the success in fulfilling a system's purpose strongly depends on the selected configuration parameters, i.e., its hyperparameters. Example domains are data cleaning, machine learning, pattern mining, anomaly detection and many more. For all such tasks, picking appropriate hyperparameter values is crucial for achieving the desired outcomes. A duplicate detection run, for instance, with non-optimized comparison metrics, similarity thresholds, window sizes etc. will not be able to find accurate duplicates. With modern systems getting more and more complex and machine learning becoming an ever more popular approach, we even see a trend towards an increasing number of parameters that are subject to tuning. For the tuning process, numerical hyperparameters of data type *float* and *integer* are especially hard to optimize due to their infinite parameter space. To find a well-performing set of hyperparameters, non-domain experts need to either explore the parameter space manually or rely on parameter search procedures, such as *Bayesian Optimization* [31, 36, 14, 12] or *Grid Search* [24, 20]. A manual exploration usually involves many iterations of applying the to-be-parameterized method to a set of labelled data, reviewing the performance and subsequently adapting the method's parameters. In contrast, existing automated parameter search procedures significantly reduce the manual work involved in this process, but they are still limited by the available amount of labelled data; identifying and labelling large datasets is a manual and time-intensive, thus costly, task. We therefore propose a novel hyperparameter optimization framework that learns not only promising default parameters but also parameter dependencies that enable easy parameter adjustments to new, unlabeled and yet unseen datasets.

We found algorithms for anomaly detection on time series data to be a class of algorithms that is particularly sensitive to parameter changes; moreover, the amount of high-quality labelled data available is very small [35]. To demonstrate our work's capabilities, we examplarily solve the hyperparameter optimization task for a variety of time series anomaly detection algorithms. We define a *time series* as a sequence of individual measurements, such as utilization percentage in computer systems, geo-location in navigation systems, or frequency in power grid monitoring, sorted by time. We call a measurement an *anomaly* whenever the sensor value is unexpected at that point in time. The occurrence of anomalies in real-world scenarios often indicates a critical situation where an autonomous system or human must intervene to prevent fraud, failures, or aggravation of a patient's medical condition. However, the nature of anomalies is their rare occurrence and unknown distribution [18], which makes implementing reliable detection systems a difficult challenge. Various research communities came up with different anomaly detection methods, of which many are specifically designed to perform well within a certain domain. We have seen work focusing on detecting anomalies in credit card transactions

Algorithm	Family [35]	Numerical			Total
		Float	Integer	Boolean	
STOMP [51]	distance	1	2	-	3
DWT-MLEAD [42]	distribution	1	2	-	3
Series2Graph [7]	encoding	-	4	-	4
Sub-LOF [9]	distance	-	5	-	5
Donut [48]	reconstruction	1	5	-	6
Sub-IF [27]	trees	2	3	1	6

Table 1.1: Selection of anomaly detection approaches and their hyperparameters by category.

to identify fraudulent behavior [4], an approach to detect sophisticated cyberattacks in large-scale computer networks by identifying anomalous traffic patterns [23], how malignant tumors can be indicated using computer-aided diagnoses on MRI images [38], and a knowledge-free anomaly detector for the spacecraft industry, which uses sensor data readings to indicate flawed components [16]. However, applying one method for detecting anomalies across different application domains can be very challenging. Using an anomaly detection system developed for a specific application, impairing knowledge of the domain it is intended to be used in, to another domain, is not straightforward, because anomalies in one domain do not have to be anomalous in another domain. Another major issue one faces in anomaly detection, besides labelled data being rare, is the amount of noise in the data. Moreover, the contained noise might be very similar to the actual anomalies, which makes distinguishing between the two very challenging [10]. Additionally, when dealing with time series data, the definition of abnormal behavior might be domain-specific. Thus, sticking with a once-created definition of expected patterns might not be sufficient to detect anomalies in the future. We also found that many anomaly detection approaches have a decent number of hyperparameters that are subject to tuning in order to reliably detect anomalies in its application scenario. Especially with the increasingly popular usage of Deep Learning techniques in recent publications, we see a lot more hyperparameters that adjust the networks' capacities, training configurations or validation procedures [1, 48, 25, 29, 22, 43, 19, 11]. Table 5.1 shows a selection of methods and their hyperparameters by category that stem from five out of six anomaly detection families introduced by Schmidl et al. [35]. All shown detection methods contain between three to six hard-to-optimize numerical hyperparameters of type *float* and *integer*.

To ease the parametrization problem, we propose *Hyper Parameter Explanation* (HYPEX), a generally applicable hyperparameter optimization framework that provides explainable parameter rules. It uses synthetically generated datasets to determine optimal parameter values and to identify relationships between (a) different algorithm parameters and (b) algorithm parameters and dataset characteristics. Possible dataset characteristics that parameters depend on vary from one application domain to another such as number of columns or rows, number of duplicates, number of errors, or the ratio string to number values in duplicate detection; sentence lengths, type of language, number of words, or encoding in natural language processing; or time series length, base oscillation variance or frequency, anomaly length, or contamination ratio in time series anomaly detection.

HYPEX generates computation rules for all identified dependent parameters, whether they depend on another hyperparameter or a dataset characteristic, and identifies generally well-performing fixed values for independent ones. When applying HYPEX's parameter-rules to a previously unseen dataset without annotations, our parameter model uses a combination of fixed parameter values and computation rules to suggest a set of well-performing parameters. HYPEX can be used

(i) to replace manual parameter optimization for a specific use case,

(ii) to find good parameters even in the case of no or insufficient training data,

(iii) to speed up automatic parameter search by restricting the search space, and

(iv) to re-tune algorithms in environments, where the definitions of normal behavior and anomalies change over time.

HYPEX first creates synthetic training datasets, which all differ in user-defined data characteristics. The influence of these characteristics on the hyperparameters is one aspect that shall be explored. The set of data characteristics with their different settings represents possible variations of incoming real-world data, e.g., base oscillation frequencies, variances, encodings, or data formats. Given the generated datasets, HYPEX uses Bayesian optimization to identify optimal hyperparameters for each of the generated datasets. It then distils the dependence between the data characteristics and the optimal hyperparameters into a causal parameter model. This parameter model can be used to automatically predict well-performing parameters on yet unseen real-world data. In summary, this thesis makes the following contributions:

(i) Dataset generation with mutation rules to explore certain numeric dataset characteristics (Section 4.1).

(ii) Distributed execution of very many Bayesian optimization tasks (Section 4.2).

(iii) Identification of causal dependencies between numeric dataset characteristics and hyperparameters as well as between individual hyperparameters (Section 4.3).

(iv) Identification of fixed default values for data-independent parameters (Section 4.4).

(v) Evaluation of final parameter model and comparison to default parameters as well as heuristics proposed by other authors in the context of time series anomaly detection (Chapter 5).

2 Related Work

Optimizing a system's or machine learning (ML) model's hyperparameters is a well-known task in many research areas. The three most common approaches for this task are (i) *Random Search* [6], (ii) *Grid Search* [24, 20], and (iii) *Bayesian Optimization* [31, 36, 14]. All three algorithms optimize hyperparameters in a way that a user-defined optimization criterion, such as F1 score, AUC-ROC score, or accuracy score, is maximized on a given dataset. While *Random Search* uniformly samples from a given parameter distribution until its time constraint is met, *Grid Search* tests all parameter combinations given by a user-defined parameter grid. *Bayesian Optimization* uses a feedback loop to learn from previous parameter evaluations and improve its parameter suggestions over time (see Section 3.5 for more details). However, all three algorithms require large amounts of labelled data to evaluate the algorithm or ML model, which is subject to optimization. Our approach overcomes this limitation by generating synthetic, labelled datasets and then transferring learned parameter dependencies. The research community has also come up with systems specifically designed to optimize hyperparameters in the context of anomaly detection. We now discuss two such systems, namely *Opprentice* and *Isudra*.

Opprentice *Opprentice* [26] is a novel approach using supervised machine learning to improve the quality of anomaly detection in practice. Their work focuses on removing the manual work required to adjust parameters and thresholds to reliably detect anomalies. *Opprentice* applies multiple existing anomaly detectors to the incoming data in parallel while collecting all detector's outputs, i.e. anomaly scores. Domain experts are required to label anomalies in the incoming real-world data on a regular basis. The combination of the detectors' outputs and the manually generated labels are used to train a random forest classifier to find reliable detector parameters and thresholds. The authors show that their system removes the manual iterative parameter and threshold tuning. However, domain experts are still required for data labelling purposes.

Isudra Dahmen et al. propose Isudra [12], an indirect supervision approach for anomaly detection methods. *Isudra* was developed in the context of detecting anomalous data points in clinical health data to optimize and tune existing unsupervised anomaly detectors to a concrete application setting. Their approach requires clinicians to label sensor time series data with detected health events. The labelled data gets decomposed into smaller sub-sequences using the sliding window approach with window size ws. From each of the resulting windows w, Isudra extracts descriptive features $fs(w)$ and, then applies an anomaly detector D with a specific set of detector hyperparameters to these features. Once the anomaly detector terminates, the Isudra supervisor calculates a score by comparing the detected anomalies with the ground truth data. Subsequently, Bayesian optimization is used to identify the most effective configuration of window size ws, feature set fs,

unsupervised anomaly detector D and detector hyperparameters $\mathcal{H}$. The authors show that the indirect supervision approach delivers significantly better performance than their comparative methods iForest and One-Class SVM in detecting six out of seven health events. However, while Isudra automatically optimizes detector parameters, it still requires domain experts, i.e. clinicians, to label a significant amount of anomalous events and, in this way, generate ground truth data.

Our approach also uses the automated hyperparameter optimization technique Bayesian optimization. However, we overcome the strong requirement of domain experts providing anomaly labels by using a data generator to generate synthetic ground truth data with anomaly labels attached. Moreover, our work returns a set of parameter-rules, which can be used to adjust a detector's parameters on a new dataset.

3 Tuning Time Series Anomaly Detectors

In this section and throughout the paper, we focus on the hyperparameter optimization of time series anomaly detection algorithms, because hyperparameter optimization for this specific domain of algorithms poses a particular difficult task. Many anomaly detection algorithms require good parameter choices, despite the fact that high-quality training data is very rare [35]. We first define time series data, possible anomaly types in time series data, the process of anomaly detection, and how to assess a time series anomaly detection algorithm. Then, we introduce GutenTAG [47], a tool to generate time series datasets with anomalies. With GutenTAG, we are able to create synthetic datasets with well defined, reliable anomaly labels and known, controlled characteristics for our parameter-rule-discovery. Finally, we present the hyperparameter optimization algorithm Bayesian Optimization [31, 36, 14, 3]. Our parameter optimization system HYPEX uses Bayesian Optimization to fine-tune the hyperparameters of anomaly detectors on synthetic datasets generated by GutenTAG.

3.1 Time Series Data

We define a time series T as an ordered set of data points T_i: $T = \{T_0, T_1, ..., T_{n-1}, T_n\}$. We call a time series with single attribute data points univariate time series. Our approach only considers data points of real-valued data type, i.e., float, integer, or boolean.

3.2 Time Series Anomalies

Following related work [2], we identify three different types of anomalies in time series data: (i) point anomalies, (ii) contextual anomalies, and (iii) collective anomalies. We call an anomaly a *point anomaly* if a single data point can be identified anomalous with respect to the other data points in the series. An anomaly is classified as a *contextual anomaly* if a data point gets interpreted as anomalous only within certain contexts. Therefore, any data instance's attributes are sub-divided into two groups: contextual and behavioral attributes [10]. Consider a time series dataset containing the measured amounts of rain in an arbitrary but fixed city. We know that the average amount of rain falling over the year strongly depends on the current season. While there is not much rain in the summer months on average, one expects higher amounts of rain in the winter months. Thus, whether a certain data point in the described time series dataset has to be interpreted as anomalous, depends on the contextual attribute, the measurement timestamp. Contextual anomalies are well known from time series [45, 34] as well as spacial datasets [28, 37]. A set of data points is called a *collective anomaly* whenever the contained data points on their own are not interpreted as anomalous behavior, but their collective occurrence is.

Consider a time series dataset containing the measured amounts of rain in summer. While a rainy day itself must be seen as usual behavior, the collective occurrence of many rainy days, e.g., over two weeks, is unusual, thus anomalous. Collective anomalies are well known from sequence [44, 41], graph [30] and spacial datasets [37].

3.3 Time Series Anomaly Detection

We define an unsupervised anomaly detector as a black-box predictor which takes a time series T as input and outputs a score $s \in \{0, 1\}$ for each data point $T_i \in T$. A score s indicates the detector's confidence that a data instance T_i is anomalous. Various research communities have come up with such anomaly detection techniques. Wenig et al. [47] propose a grouping into six base methods (i) forecasting, i.e. learning a model to predict several upcoming time steps based on a context window and comparing the predicted values with the observed ones, (ii) reconstruction, i.e. encoding time series subsequences of normal behavior into a latent space and reconstructing test values from that latent space to compare with the observed ones, (iii) encoding, i.e. mapping subsequences of a time series into a latent space and computing anomaly scores directly from the low dimensional representation, (iv) distance, i.e. measuring the distance between time series points or subsequences where smaller distances exist between normal observations and larger distances to anomalous observations, (v) distribution, i.e. tracking distributions over data points, subsequences or by windowing and determining anomalous behavior by frequency, and (vi) isolation tree, i.e. partitioning data points or subsequences by building an ensemble of random trees and judging abnormality based on average path length in the trees to the root nodes where a shorter path is an indication of anomalous behavior. To classify data instance as anomalous, we need to turn the detectors' confidence scores into binary labels by setting a threshold. All data instances whose scores exceed that threshold are seen as abnormal. This threshold significantly influences the anomaly detector's result quality. To eliminate a potential tuning of the threshold parameter, we use the *AUC-PR* score [8, 13] as a performance measure for anomaly detector outputs. This measure is especially popular in applications dealing with learning on imbalanced data. Anomalies fulfil this criterion as they are rare events by nature.

3.4 Generating Time Series with Anomalies

GutenTAG [1] is an extensible time series anomaly data generator proposed by Wenig et al. [47, 46]. Each generated time series has a base oscillation behavior. GutenTAG comes with the following existing base oscillations: (i) sine, (ii) electrocardiogram (ECG), (iii) random walk, (iv) cylinder bell funnel, and (v) polynomial, but it can be enhanced with additional ones. On top of the chosen base oscillation, we can further define a general trend within the time series. Here, (i) sine and (ii) polynomial are most commonly used. The user can decide to inject different anomalies into the time series. GutenTAG comes with nine pre-defined anomaly types: (i) amplitude, (ii) extremum, (iii) frequency, (iv) mean, (v) pattern, (vi) pattern shift, (vii) platform, (viii) trend, and (ix) variance. Such

[1] Code available at `https://github.com/HPI-Information-Systems/gutentag`

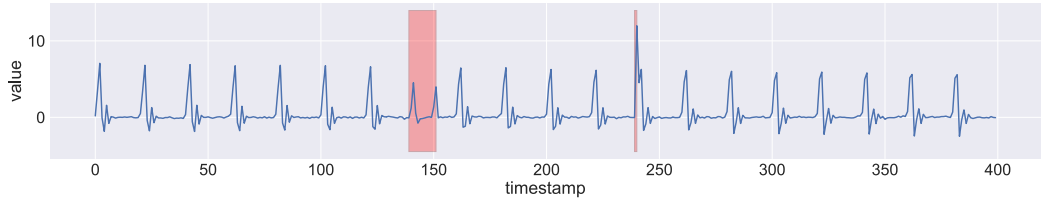

Figure 3.1: Exemplary time series generated with GutenTag [46].

anomalies can either be placed at certain exact positions, or we can let GutenTAG decide where to exactly place the anomalies. Their algorithm ensures that anomalies do not overlap. Each generated time series dataset contains labels for the injected anomalies, and GutenTAG provides access to the generation metadata. This information is used to extract and control the dataset characteristics. Figure 3.1 shows a univariate real-valued anomaly time series generated by GutenTAG where the underlying base oscillation simulates ECG time series data. Two anomalies were added to the data, both indicated by the red background color: the first at the timestamp *140* of type *frequency* and length *12*, the second at position *240* of type *extremum* and length *1*. We generate anomalous time series data with GutenTAG to optimize detector hyperparameters and discovery parameter rules by identifying dependencies between data characteristics and optimized hyperparameters.

3.5 Hyperparameter Optimization Algorithms

Almost every anomaly detector can and needs to be adjusted via a set of hyperparameters. While some parameters, such as thread counts and memory limits, tune solely runtime metrics of the algorithms; other hyperparameters, such as capacities, context window sizes, similarity thresholds, learning rates, or distance measures, tune the detector's performance and are, therefore, much more critical. So to achieve a consistent, reliable detector performance on observed time series data, its hyperparameters must be optimized. The three most widely adopted hyperparameter optimization algorithms are *Random Search*, *Grid Search* and *Bayesian Optimization*. All three have in common that they optimize the parameters w.r.t. a user-defined criterion, e.g., F1-Score, or accuracy score. While *Random Search* and *Grid Search* simply return the best performing hyperparameters after testing a bunch of parameter configurations, *Bayesian Optimization* bases hyperparameter guesses on past parameter evaluations. *Bayesian Optimization* is commonly used when optimizing an expensive to evaluate objective function f [15], e.g., tuning an ML model's architecture, which involves repeatedly fitting the adjusted model on train data and subsequently evaluating it on test data. Especially model training can be a very time-consuming and, thus, expensive procedure. Bayesian Optimization appears to be an effective approach for reducing the required number of tested model architectures to find a suitable one. The most widely adopted Bayesian optimization method is Sequential Model-Based Optimization (SMBO). Instead of optimizing the objective function f directly, SMBO uses a probabilistic model $P(f(\mathcal{H}) \mid \mathcal{H})$ as a surrogate for f [14]. Until a time constraint $\mathcal{C}$ is met, SMBO keeps iterating with each iteration consisting of the following four steps: (i) updating the probabilistic model based on previously collected benchmark results, (ii) selecting the next

best guess parameters $\mathcal{H}$ based on the probabilistic model, (iii) evaluating the objective function f with the parameters $\mathcal{H}$, which is the most expensive step, and (iv) collecting as well as saving the benchmark results $(\mathcal{H}, f(\mathcal{H}))$ for the upcoming optimization steps. The research community came up with several different samplers to generate the next best parameter guess based on the previous evaluated parameters [5]. Our approach uses the Tree-structured Parzen Estimator (TPE) algorithm. In each iteration, it fits two Gaussian Mixture Models (GMMs) per hyperparameter $\eta \in \mathcal{H}$, the first GMM $l(\eta)$ on the set of well performing hyperparameters $f(\eta) > y^*$ and the second $g(\eta)$ on the remaining ones $f(\eta) \leq y^*$. The split value y^* is automatically chosen by the TPE algorithm to match some quantile γ of the observed values for the optimization criterion. In each iteration, for each hyperparameter, TPE chooses the value η that maximizes the ratio $l(\eta)/g(\eta)$. Finally, of all the evaluated parameter guesses $\mathcal{H}$, the one with the best performance score $f(\mathcal{H}))$ is returned. Our work builds upon the open-source SMBO framework Optuna [3], which implements the TPE algorithm with its *TPESampler*.

4 Finding Hyperparameter Explanations

In this section, we propose HYPEX, a framework to automatically optimize time series anomaly detector hyperparameters and extract parameter rules without requiring access to manually labelled ground truth data. We use a fully controlled data environment to gain insights on causal dependencies between numerical data characteristics and well-performing parameter sets, as well as relations between hyperparameters themselves. Our procedure consists of five consecutive steps (i) time series data generation with anomaly injection (Section 4.1), (ii) distributed hyperparameter optimization (Section 4.2), (iii) parameter-rule-discovery by finding causal dependencies between data characteristics and detector's hyperparameters (Sections 4.3.1 to 4.3.3) coming up with a set of parameter model candidates (Section 4.3.4), (iv) validation of parameter model candidates resulting in a final parameter model incorporating all significant parameter rules (Section 4.3.5), and (v) identification of data-independent, fixed parameter values for those parameters not covered by the parameter model (Section 4.4). The resulting final parameter model and fixed parameters, can be used to calculate future hyperparameters on unseen time series data based on the data's characteristics. Figure 4.1 shows a high-level architecture overview of our HYPEX approach. All time-intensive tasks, namely steps (i), (ii), (iv) and (v), are executed in a distribution manner on a Dask [33] cluster utilizing multiple worker nodes to speed up the overall run time. This section explains HYPEX's phases from top to bottom.

4.1 Data Generation

We use the time series anomaly data generator GutenTAG [47, 46] to generate the synthetic datasets, based on which we optimize the anomaly detector's parameters. Each physical node in the Dask cluster performs the data generation task. To test how an anomaly detector's parameters need to adapt w.r.t. a change in a specific data characteristic, we introduce *time series mutations*. We define a time series mutation as an attribute change in the GutenTAG configuration used to generate the synthetic time series data. A generated dataset may contain $n \in \mathbf{N}^+$ time series mutations, so-called *mutation sets*, thus incorporating simultaneous changes in different data characteristics. HYPEX uses user-defined mutation sets to generate synthetic datasets that vary in an arbitrary but fixed set of data characteristics. Figure 4.2 shows an example of a mutation set that contains a single time series mutation of the attribute *base-oscillation.frequency* applied to an arbitrary but fixed, predefined time series *ecg-data*. In this case, HYPEX uniformly samples values for the attribute *base-oscillation.frequency* of the user-provided time series *ecg-data* from the provided range $[10, 50]$. The sampling creates a total of 10 mutated GutenTAG configurations. Subsequently, those 10 different GutenTAG configurations are used to generate the actual 10 datasets.

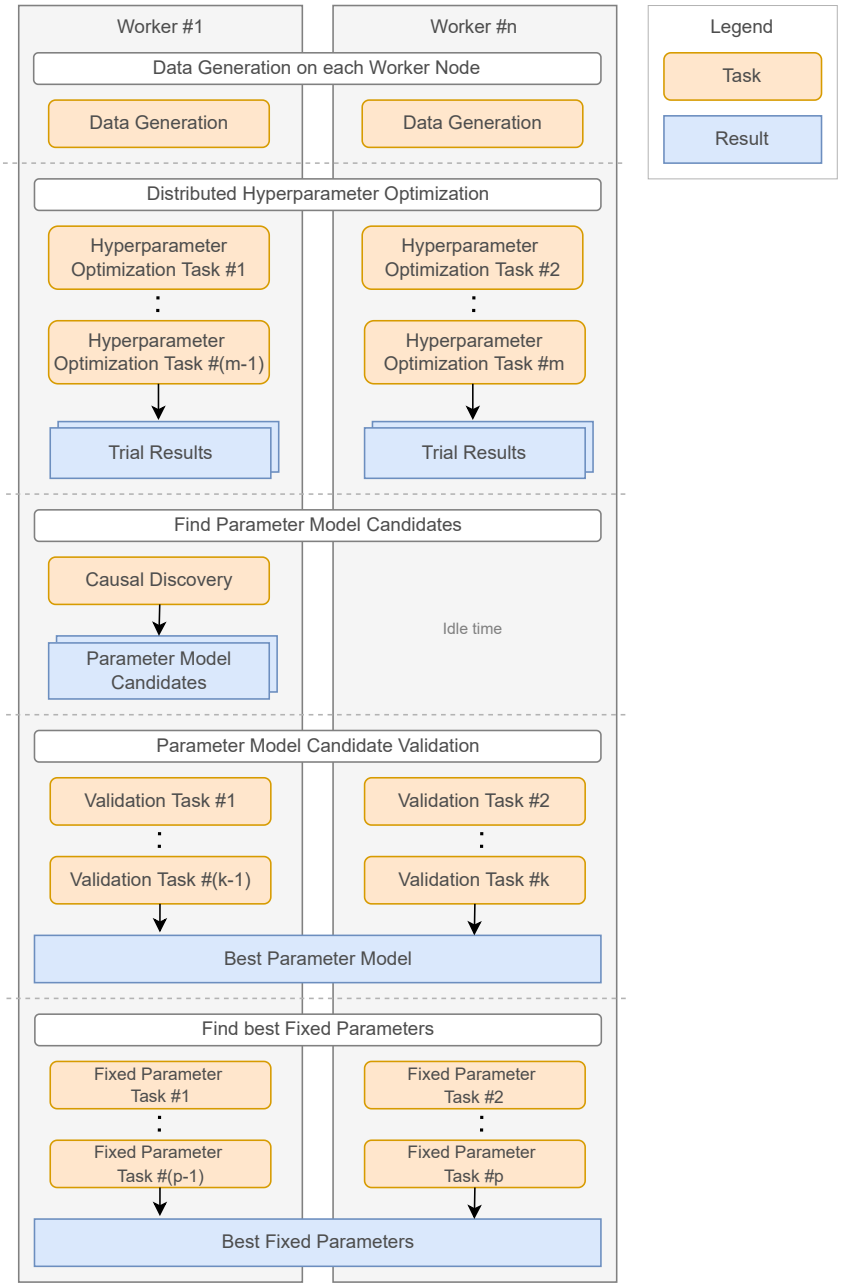

Figure 4.1: Control flow of a distributed HYPEX execution consisting of five major execution phases: (i) data generation (Section 4.1), (ii) distributed hyperparameter optimization (Section 4.2), (iii) parameter-rule-discovery (Section 4.3), (iv) parameter model validation (Section 4.3.5), and (v) fixed parameter identification (Section 4.4).

```yaml
name: ecg-data # The time series to mutate
n_samples: 10 # The number of mutation sets to generate
mutations:
  - paths:
      - base-oscillation.frequency # The attribute to mutate
    dtype: int
    min: 10
    max: 50
```

Figure 4.2: Exemplary mutation of the data characteristic *base-oscillation.frequency* of a user-defined time series called *ecg-data*.

To conclude the data generation, HYPEX splits the set of generated time series datasets into a train, validation and test set. We assign 60% of the time series datasets to the train and 20% each to the validation and test sets.

4.2 Hyperparameter Optimization

Once the data generation completes, HYPEX enters the distributed hyperparameter optimization. For each time series dataset in the training split that GutenTAG generated, we independently optimize the given anomaly detector's parameters such that the algorithm performs well on the selected dataset. We use the *AUC-PR* score [8, 13] as the trials' optimization criterion with the open source Bayesian optimizer Optuna [3]. We define a trial as a monitored algorithm run with a suggested hyperparameter configuration on one of the synthetic time series dataset. HYPEX keeps track of the performed trials which includes the tested hyperparameters and the resulting performance scores. The number of trials this approach requires to produce reliable results strongly depends on the number and data type of the hyperparameters to optimize.

Task Distribution HYPEX speeds up the optimization procedure using distributed computing techniques: It splits the workload into smaller sub-tasks, which can be computed independently and in parallel on potentially different physical nodes. For the implementation of this distribution, we have chosen the framework Dask [33]. Dask provides dynamic task scheduling as well as a data collection library - both accessible through a convenient Python API. We wrap each optimization trial into a Dask task such that we can submit the entire set of tasks to the Dask scheduler at once. Dask then controls and schedules the tasks on the cluster's worker nodes. This procedure requires us to ensure that each trial can run on each included physical node, i.e., the synthetic data must be present on all nodes. HYPEX therefore seeds the random number generator in GutenTAG such that all physical nodes generate all and the exactly same input datasets. To ensure the Bayesian optimizer bases its parameter suggestion on previous parameter evaluation runs, each trial requires access to not only the trial runs on the same physical cluster node, but to all trials from all cluster nodes. Optuna achieves this trial synchronization by storing trial results in a MySQL database, which HYPEX launches on the Dask scheduler

on start up. Each trial first connects to the database to fetch previous trials' results, then suggests a new set of hyperparameters, runs the desired algorithm with the suggested hyperparameters, computes the AUC-PR score from the algorithm's anomaly scores, and finally persists the tested parameters and AUC-PR score in the MySQL database.

4.3 Parameter Rule Discovery

We use the optimized hyperparameters found by Optuna (Section 4.2) to discover rules on (i) how hyperparameters depend on selected data characteristics represented by the applied time series mutations and (ii) how hyperparameters depend on each other. In this section, we first present a de-noising step for the trial results data in which HYPEX later on searches for parameter-rules (Section 4.3.1). Then, we guide through the causal skeleton estimation (Section 4.3.2) and introduce the causal edge orientation (Section 4.3.3). We discuss how HYPEX compiles the identified parameter rules into a parameter model, which is used later in the process to predict hyperparameters on new, unseen time series dataset based on specific data characteristics (Section 4.3.4). Finally, we present an approach to validate discovered parameter rules despite a high uncertainty of the amount of remaining noise in our data (Section 4.3.5).

4.3.1 Noise Reduction

The set of collected Optuna trials consists of parameter configurations with different performance AUC-PR scores. However, we only want to find parameter-rules in the trial results that can predict well performing hyperparameters. All trials with low AUC-PR scores are thus treated as noise. To remove a large portion of that noise, we filter out all trials with AUC-PR scores lower than a dynamic threshold γ, which we calculate for each time series in the training split: We define the time series' threshold γ as the top 10% AUC-PR score quantile. To improve our parameter-rules' explainability, we avoid finding parameter-rules for parameters with low impact on an anomaly detector's performance. HYPEX uses the parameter importance evaluator fANOVA [21] to identify a detector's important parameters. fANOVA trains a random forest regressor to predict the AUC-PR scores based on a given parameter configuration. The random forest regression's feature importance is used to assign importance scores to the anomaly detector's parameters. We only consider parameters with a minimum importance of 0.01 as significant. The set of noise-reduced trials and the information on parameter importances, gets combined into a matrix M of dimensionality $C \times N$. N defines the number of trials left after reducing the noise and the number of columns $C = p + d + 1$ the number of anomaly detector parameters p with an importance score ≥ 0.01 plus the number of applied time series mutations d and an extra column for each trial's achieved AUC-PR score. Consider an anomaly detector taking two parameters *window size* with an importance score of 0.995 and *random state* with an importance score of 0.005. Moreover, the datasets contain a single applied time series mutation *base-oscillation frequency*. The resulting matrix M consists of three columns, namely *window size*, *base-oscillation frequency*, and the AUC-PR score. *random state* gets removed due to its low importance score of 0.005. The number of rows in M depends on the performed trials' AUC-PR score distribution. On the assumption of a

uniform AUC-PR score distribution and 300 trial runs, we expect M to have $0.1 \cdot 300 = 30$ rows.

4.3.2 Skeleton Estimation

To identify causal dependencies between columns of the noise-reduced data matrix M (Section 4.3.1), we perform linear and non-linear independence tests. For these tests, we found that most existing approaches for the detection of non-linear dependencies are too sensitive on our data given the potentially still very small signal-to-noise ratio. This is why we use a fairly simple, still powerful, regression-based non-linear independence test in combination with the PC algorithm [39]. The PC algorithm is one of the oldest methods to discover causal dependency graphs [39, 17]. It supports plugging in many statistical tests for checking independence and Conditional Independence (CI), which makes it usable in a variety of settings. In HYPEX, we integrate an open-source implementation of the PC algorithm in Python [2] and extend it with own regression-based independence and CI tests called Non-Linear Regression by Transformation (NLRegT) independence test and NLRegT CI test. Note that they are applicable and perform well only on parameters of the numerical data types *integer* and *float*. The PC algorithm uses the following five steps to estimate an undirected version of the true causal graph, which we call the causal graph skeleton:

(i) Create a fully connected graph G where each column in the data matrix, i.e., hyperparamater and dataset characteristic, is represented by a node in G.

(ii) For each edge $(A, B) \in G$, run the (in-)dependency test and remove it from G if the variables A and B are (unconditionally) independent. We use dynamic confidence thresholds α (for more details see Section 4.3.5).

(iii) For each of the remaining edges $(A, B) \in G$ and each set of nodes $Z = \{Z_1, ..., Z_n\}, n \in \mathbb{N}^+$ either all connected to A or all connected to B, remove the edge (A, B) if A and B are conditionally dependent under Z. Start with $n = 1$ and repeat this step with increasing set sizes n. Consider a true causal graph $A \to B \to C$. Step (ii) finds (unconditional) dependencies between $\{A, B\}$, $\{B, C\}$, and $\{A, C\}$. However, the dependency $\{A, C\}$ only gets identified because there exists a path (A, B, C) between A and C. The PC algorithm uses the conditional independence test to identify such dependencies $\{A, C\}$ and remove them from the estimated causal graph. Note, that it is possible to additionally have an edge $A \to C$ in the true causal graph. In this case, $\{A, C\}$ is not tested conditional dependent and their edge is therefore not removed from the estimated causal graph.

In the following, we explain our NLRegT independence and CI tests which HYPEX uses with the PC algorithm to estimate the causal graph.

NLRegT Independence Test Our NLRegT test is a very robust and powerful independence test for our application scenario. It runs a least squares optimization on pre-transformed data and comes with two transformations by default: *linear* and *hyperbola*. The *linear*

[2] Code available at `https://github.com/keiichishima/pcalg`

transformation in front of the least squares optimization allows testing for relationships between a predicting variable u and a predicted variable v of the form

$$v = \beta \cdot u + c$$

β being the linear regression's coefficient, and c being its intercept. The *hyperbola* transformation on the other hand, enables us to detect causal dependencies between u and v of the form

$$v = \beta \cdot \frac{1}{u} + c$$

This transformation concept is further extendable. It allows to specifically define the dependencies that we consider relevant for our parameter-rule-discovery use-case. Within the domain of anomaly detection on time series data, we found that *linear* and *hyperbola* dependencies are most common. Algorithm 1 shows the procedure of fitting the non-linear regression by transformation. It takes the slices $\mathcal{X}$ of the data matrix M containing the predicting variable and $\mathcal{Y}$ containing the predicted variable, a data series $\mathcal{W}$ containing the achieved AUC-PR scores, and a set of transformation functions $\mathcal{T}$ as input. For each of the provided transformation functions $\in \mathcal{T}$, we fit a linear regression. For this purpose, we transform the input data of the predicting variable $\mathcal{X}$ with the current transformation function (Line 5). The linear regression of $\mathcal{Y}$ onto the transformed data $\mathcal{X}_T$ gets fit by using the trials' squared AUC-PR scores $\mathcal{W}^2$ as sample weights (Line 6). We thereby increase the weight of trials with higher AUC-PR scores and decrease the weight of those with smaller score values. Afterwards, the fitted regression model itself is scored using the R^2-Score (Line 9). Finally, the model with the highest R^2-Score and the R^2-Score itself are returned (Line 13). We recall that $R^2 = 1 - \frac{RSS}{TSS}$ is defined as the difference of 1 and the residual sum of squares (RSS) divided by the total sum of squares (TSS). R^2-Scores range between $(-\infty, 1.0]$ where a score of 1.0 is achieved by a model which can explain the variance in $\mathcal{Y}$ in total.

Algorithm 1 Fit a non-linear regression by transformation.

```
1:  procedure fitNonLinearRegression(𝒳, 𝒴, 𝒲, 𝒯 = [f_linear, f_hyperbola])
2:      score_max ← 0
3:      model ← null
4:      for transform in 𝒯 do
5:          𝒳_T ← transform(𝒳)
6:          model ← fitLinearRegression(𝒳_T, 𝒴, 𝒲²)
7:
8:          Ŷ ← model.predict(𝒳_T)
9:          score ← r2Score(𝒴, Ŷ, 𝒲²)
10:
11:         if score > score_max then
12:             score_max ← score
13:             model ← model
        return score, model
```

To test for independence between $\mathcal{X}$ and $\mathcal{Y}$, we check whether the calculated R^2-Score is smaller than a provided threshold α. The quality of the causal discovery output strongly

depends on the chosen value for α. We provide more information on how we use α to generate different model candidates in Section 4.3.5.

NLRegT CI Test While the provided (unconditional) independence test is generally applicable, our conditional independence test performing $\mathcal{X} \perp \mathcal{Y} \mid \mathcal{Z}$ assumes the data to be generated under the Additive Noise Model (ANM). For this case, Peters et al. [32] showed that the conditional independence test can be mapped to an unconditional one. The ANM assumes that there is a functional relationship between $\mathcal{Z}$ and $\mathcal{X}$ such that $\mathcal{X} = f(\mathcal{Z}) + \mathcal{N}_x$ with $\mathcal{N}_x$ being a zero-mean noise independent from $\mathcal{Z}$. The same assumption applies to $\mathcal{Y} = g(\mathcal{Z}) + \mathcal{N}_y$. These assumptions allow the redefinition of $\mathcal{X} \perp \mathcal{Y} \mid \mathcal{Z}$ to $\mathcal{N}_x \perp \mathcal{N}_y$ [50].

Algorithm 2 NLRegT CI Test

```
1: procedure NLRegTCI(X, Y, Z, W, β)
2:     (model_X) ← fitNonLinearRegression(Z, X, W)
3:     ε_X ← X − model_X.predict(Z)
4:
5:     (model_Y) ← fitNonLinearRegression(Z, Y, W)
6:     ε_Y ← Y − model_Y.predict(Z)
7:
8:     (score) ← fitNonLinearRegression(ε_X, ε_Y)
9: return score > β
```

Algorithm 2 shows the application of this redefinition. This CI test uses two steps (i) estimation of the noise terms $\mathcal{N}_x$ and $\mathcal{N}_y$, and (ii) testing for independence between the two estimated noise terms $\mathcal{N}_x$ and $\mathcal{N}_y$. To estimate the two noise terms, we fit the non-linear regression defined in Algorithm 1 of $\mathcal{Z}$ onto $\mathcal{X}$ (Line 2) and $\mathcal{Y}$ (Line 5). We calculate the regressions' residuals ϵ_X (Line 3) and ϵ_Y (Line 6). To test for independence between the residuals ϵ_X and ϵ_Y, we once again fit a non-linear regression to check whether its R^2-Scores is larger than a threshold β. HYPEX evaluates the different β threshold values 0.2, 0.4, 0.6, and 0.8. The best performing among these gets selected (for more details see Section 4.3.5).

4.3.3 Edge Orientation

Up until now, we found correlated parameter-parameter and parameter-characteristic edges as undirected relationships. Though, to derive parameters from other parameters or dataset characteristics, these relationships need to be oriented. HYPEX's edge orientation consists of three consecutive steps (i) incorporation of prior knowledge on dataset characteristics, (ii) Completed Partially Directed Acyclic Graph (CPDAG) estimation procedure from the PC algorithm, and (iii) CPDAG to Directed Acyclic Graph (DAG) conversion. In step (i), we incorporate a task-specific constraint: Because we aim to find good hyperparameters given certain dataset characteristics, edges semantically need to point from characteristics to parameters. Given the edge (A, B) with orientation $A \rightarrow B$, HYPEX removes all edges (A, B) from the graph skeleton where the node B represents one of the dataset characteristics. Step (ii) executes the PC algorithm's CPDAG estimation. The PC algorithm is guaranteed to converge to the Markov Equivalence Class (MEC) under the

causal Markov condition and faithfulness assumption and when there is no undiscovered confounder [17]. Consider, for example, a true causal graph G containing only two nodes A and B with a single undirected edge $\{A, B\} \in G$. Here, the PC algorithm identifies A and B as dependent on one another, but it cannot decide the dependency direction. Therefore, the resulting graph contains an undirected edge between the nodes A and B. Such a graph that represents the MEC and possibly contains a mixture of directed and undirected edges is called a CPDAG. To estimate the CPDAG, the PC algorithm executes the following two steps [17]:

(i) Search for *v-structures* and orient edges accordingly. A *v-structure* is a triple of nodes (A, B, C) such that there exist the undirected edges $\{A, B\} \in G$ and $\{B, C\} \in G$, but $\{A, C\} \notin G$ and the node B is not contained in the set $Z = \{Z_0, ..., Z_n\}$ under which A and C were tested conditionally independent. The edges in such a *v-structure* are oriented $A \to B$ and $C \to B$.

(ii) Use *orientation propagation* to orient possibly many of the remaining edges. To do so, search for a triple of nodes (A, B, C) such that there exists a directed edge $(A, B) \in G$, an undirected edge $\{B, C\} \in G$, but no edge between A and C. In each of the found triples, the undirected edge $\{B, C\}$ gets oriented $B \to C$.

While the resulting CPDAG might still contain undirected edges, our parameter model requires all discovered parameter-rules to have a clear orientation. Therefore, HYPEX must convert the estimated CPDAG to a DAG by removing possibly remaining undirected edges. To ensure this requirement, step (iii) in our edge orientation procedure performs a colored depth-first search [40, 49]. This removes back edges in the graph, thus breaking existing cycles and turning the CPDAG into a DAG with no undirected edges or cyclic dependencies remaining.

4.3.4 Parameter Model

HYPEX's parameter models are used to predict any number of anomaly detector parameters directly from the set of data characteristics. Furthermore, a parameter model can capture relationships between two or more anomaly detector parameters. HYPEX creates the parameter model from the dependency DAG (see section 4.3.3). For each node, HYPEX identifies all predecessor nodes and fits the NLRegT model once again using the trials' squared AUC-PR score as the sample weight. While the NLRegT model previously contained just a single feature, which was used to predict the target variable, the number of features here depends on the number of predecessors in the graph. The parameter model stores at most one NLRegT model for each anomaly detector parameter. When it comes to predicting hyperparameters on unseen data, we iterate over the estimated DAG in topological order. In each step, HYPEX uses the stored NLRegT model to predict the respective parameter based on all previously estimated parameters and data characteristics. In Section 4.4, we show how HYPEX assigns fixed values to the remaining hyperparameters that are not covered by the parameter model.

4.3.5 Parameter Model Selection

Our experiments show that the optimal α and β threshold to choose for the parameter-rule-discovery (Section 4.3.2) is algorithm and base oscillation specific. Therefore, we run the parameter-rule-discovery for different α and β thresholds leaving us with a set of parameter model candidates. The set of α thresholds to test is determined by fitting a non-linear regression on each pair of variables in the data matrix M, rounding the regressions' R^2-Scores to two decimal places while only considering those R^2-Scores ≥ 0.05 significant. The set of β thresholds is fixed and set to $\{0.2, 0.4, 0.6, 0.8\}$. For each unique combination of α and β thresholds we create a parameter model candidate by running our parameter-rule-discovery using the respective threshold values. Subsequently, we measure the parameter model candidates' performances on the validation data split. We recall that the validation split contains 20% of the generated time series datasets (Section 4.1). To test a candidate's performance, we use the parameter model to predict the anomaly detector's parameters based on data characteristics. All parameters that are not covered by the parameter model candidate are filled up with uniformly sampled random values. For each pair of model candidate and time series dataset, we independently sample non-covered parameters 10 times and rank the anomaly detector's outputs using the mean AUC-PR score across all random draws. Once all models have been benchmarked, we choose the parameter model with the highest mean AUC-PR score across all conducted tests.

4.4 Fixed Parameters

Our parameter models are expected to cover only such anomaly detector hyperparameters that depend on either a data characteristic or another hyperparameter. For the remaining, not covered, and data-independent anomaly detector hyperparameters, we identify generally well-performing, fixed values. To find these values, HYPEX again uses the Bayesian optimizer to optimize the mean AUC-PR score across all generated validation time series datasets. For each time series, we utilize the parameter model to predict the data-dependent hyperparameters based on the contained time series anomalies. Only the data-independent parameters are subject to optimization in this final step. Eventually, the best performing trial's parameters are selected as generally applicable, fixed parameters for the tested anomaly detector. The final parameter model holds a combination of NLRegT models and fixed parameters, thus being able to predict all anomaly detector parameters on unseen time series data.

5 Evaluation

In this section, we focus on the evaluation of our approach on a variety of anomaly detection algorithms and time series datasets. We start with the explanation of the experimental setup (Section 5.1), then compare the performance scores achieved by our parameter suggestions with relevant alternative approaches (Section 5.2), review HYPEX's sensitivity to automatically chosen thresholds (Section 5.3), and show identified parameter dependencies across four base oscillations and six algorithms (Section 5.4).

5.1 Experimental Setup

We evaluate our approach on six anomaly detection algorithms and four time series dataset groups containing different base oscillations and anomaly types. The included base oscillation behaviors are (i) sine, (ii) ECG, (iii) random walk, and (iv) cylinder bell funnel [46]. Each base oscillation is considered separately as we find this to have a large impact on possible parameter rules and algorithm behaviors. All generated time series datasets consist of $10,000$ individual data points each and contain 3, 6, or 9 same-length anomalies at different positions of types (i) variance, (ii) frequency, or (iii) pattern [46]. We apply time series mutations to (a) the base oscillation frequency (only applicable for base oscillations sine and ECG), (b) the base oscillation variance, (c) the length of included anomalies, and (d) the number of anomalies to include. For each of the four dataset groups characterized by the four base oscillation behaviors, we incorporate 50 time series mutations. We recall that 20% of the generated datasets containing specific time series mutations are reserved for evaluation purposes only (Section 4.1). We evaluate our approach on algorithms from five out of six anomaly detector families defined by Schmidl et al. [35]. The algorithms stem from the areas (i) distance, STOMP [51] and Sub-LOF [9], (ii) distribution, DWT-MLEAD [42], (iii) encoding, Series2Graph [7], (iv) reconstruction, Donut [48], and (v) trees, Sub-IF [27]. Each algorithm has between 3 to 6 hyperparameters subject to optimization (Table 5.1).

As we do not have any information on the true relationships between data characteristics and parameters for any of the anomaly detectors on which we evaluate our framework HYPEX, we measure the quality of our parameter model and fixed parameter suggestions by comparing them to (a) the algorithms' default parameters, (b) the manually tuned parameter recommendations proposed by Schmidl et al. [35], and (c) the optimization results achieved by the Bayesian Optimizer Optuna *(full optimization)*. While each detector's default parameter configuration stays constant no matter which time series it runs on, our parameter model as well as the TimeEval heuristics use specific data characteristics to adapt a selected subset of the hyperparameters to the time series input while keeping others constant. The *full optimization*, however, can unfold its full potential by tuning every single parameter to the specific input time series. Thus, we see the achieved performance

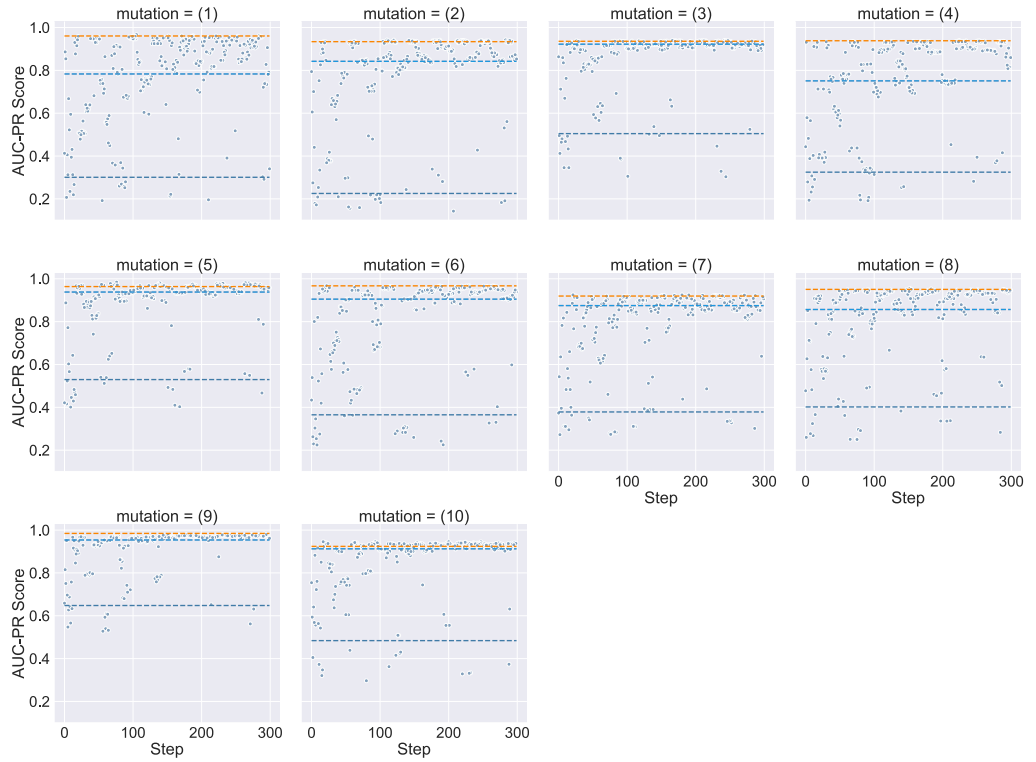

Figure 5.1: Evaluation of Donut [48] on 10 mutated sine time series datasets comparing ■ our parameter model with ■ Donut's default parameters, ■ the TimeEval heuristics, and ■ a full optimization run.

scores when using the algorithm's default parameters as a lower bound and the full optimization's best score as the upper bound for our approach's performance. We continue to use the AUC-PR score to measure the algorithms' detection quality.

5.2 Parameter Model Performance

First, we show a single anomaly detector's performance using HYPEX's suggested hyperparameters. Then, we present a general overview of the performance achieved on the tested algorithms and base oscillations.

Figure 5.1 visualizes our evaluation results on the anomaly detector Donut [48] and 10 time series datasets with base oscillation *sine* containing various time series mutations. It compares ■ our approach's performance with the performance achieved by (i) ■ the detector's default parameters, (ii) ■ TimeEval's heuristics, and (iii) ■ a full Bayesian optimization run *(full optimization)*. While the default parameters, the TimeEval heuristics, and our parameter model suggest a single hyperparameter configuration per time series dataset, the full optimization is granted 300 trials to optimize Donut's hyperparameters for each of the 10 time series datasets separately. Though, we find that parameters suggested by TimeEval and our parameter model achieve high AUC-PR scores. HYPEX's

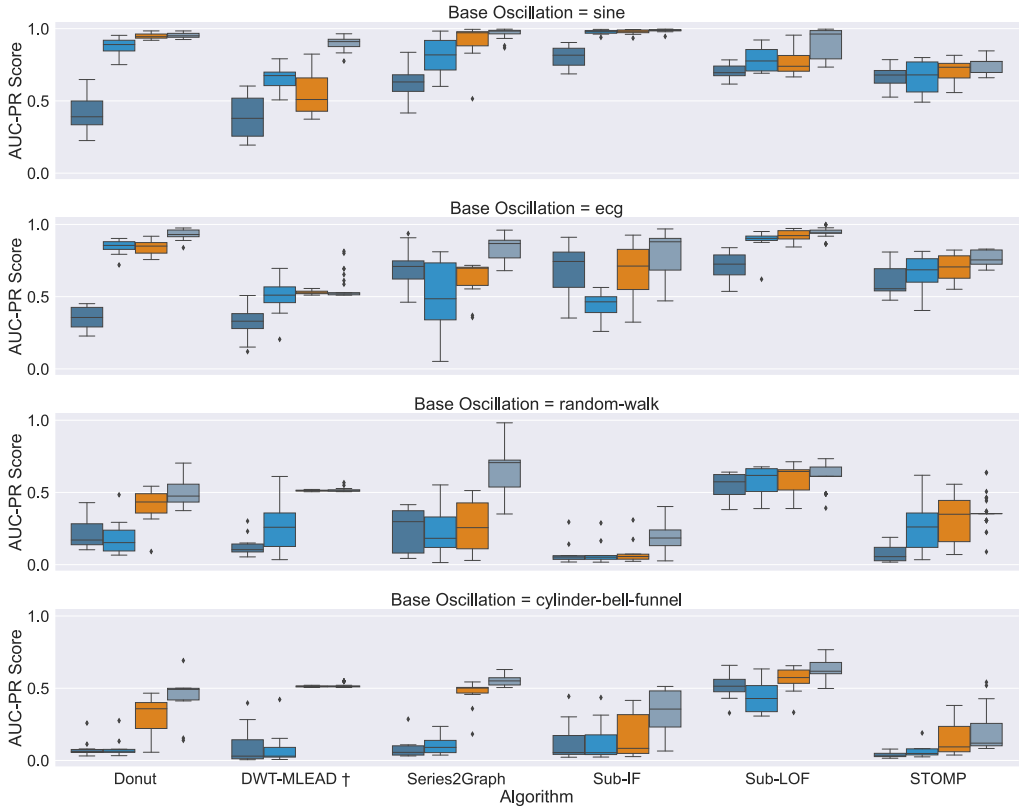

Figure 5.2: Distribution of maximum AUC-PR scores achieved by *Default Parameters*, *Timeeval Heuristics*, our *Parameter Model*, and *Full Optimization* per base oscillation and algorithm. † Empty parameter model for base oscillations *random walk* and *cylinder-bell-funnel*.

parameter suggestions' performance scores get even close to the maximum scores we see when applying the Bayesian optimizer while not requiring the input time series dataset to contain anomaly labels to optimize the parameters for that specific time series. Furthermore, we find that Donut's default parameters perform significantly worse on the evaluation datasets than the other three approaches TimeEval, parameter model, and full optimization.

We summarize our evaluation results across all six tested algorithms on four base oscillations in Figure 5.2. A single box plot shows the distribution of 10 AUC-PR scores in total. Each score represents the evaluation result on a single evaluation time series dataset. To represent the full optimization trials, we pick the maximum AUC-PR scores per time series dataset obtained in each of the 300 trial runs. Across all algorithms and base oscillations, our assumption to treat the full optimization's results as an upper bound AUC-PR score seems confirmed. The algorithms' default parameters, on average, achieve the lowest performance scores in the majority of the conducted experiments. The TimeEval heuristics mostly deliver higher performance scores than the algorithms' default parameters. However, our experiments show that they fail to predict well-performing hyperparameters

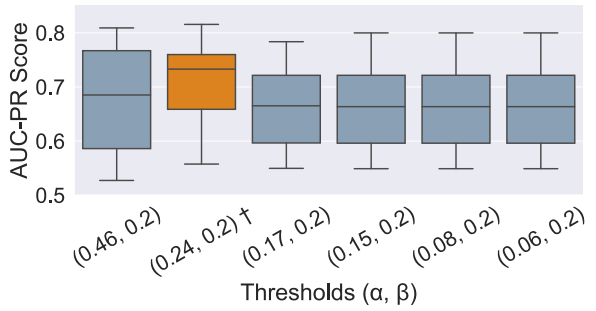

Figure 5.3: Comparison of AUC-PR score distributions by selected α and β threshold values on 10 evaluation datasets for algorithm STOMP [51] and base oscillation sine. Each named threshold tuple represents a set of tuples that all result in the identification of identical parameter-rules. † The threshold value HYPEX automatically chose (see Section 4.3.5).

in some cases, such as Donut on base oscillation random walk, Series2Graph on base oscillations ECG and random walk, and Sub-IF on base oscillation ECG.

Our parameter models' suggestions surpass the TimeEval performances in most experiments while being a fully automated approach that does not require any knowledge on the anomaly detectors' internals. The suggested fixed parameter values still outperform the detector's default parameters even in the case where HYPEX did not discover any data-dependent parameter rules for an anomaly detector, such as DWT-MLEAD on base oscillations sine, random walk, and cylinder bell funnel.

In terms of absolute performances, we see a propensity across all evaluated anomaly detectors to detect anomalies on cyclical base oscillations, such as sine and ECG, more reliable than on not cyclical ones like random walk and cylinder bell funnel.

5.3 Sensitivity to Thresholds

HYPEX uses two thresholds, namely α and β, to adjust the (in-)dependence and CI tests' minimum confidence scores. It determines the optimal values for α and β automatically during the parameter model selection (see Section 4.3.5). While HYPEX performs the optimal threshold determination on the validation datasets, Figure 5.3 shows the parameter model results on the evaluation datasets. We optimized each parameter model's fixed parameters independently for each of the threshold tuples. Many experiments show that the full optimization trial runs have low variances in optimal parameter values across different evaluation datasets. Thus, parameter-rules do not showcase their full potential as optimized fixed parameters achieve similar high performance scores. However, the full optimization's best parameter value suggestions for the algorithm STOMP on the base oscillation sine showed high variances across the different datasets. The (α, β) threshold tuple $(0.46, 0.2)$ resulted in an empty causal graph, thus the suggested parameters on the evaluation datasets are based on fixed values only. It though achieves the second best detection results on average among the compared parameter models. We also see that using fixed values only comes at the cost of increased variance across the evaluation

datasets. HYPEX automatically chose the threshold values $(0.24, 0.2)$. The resulting parameter model, which contains a mixture of parameter-rules and fixed parameter values, on average, performs best on the evaluation datasets and shows to have a lower variance than using fixed parameter values only.

5.4 Hyperparameter Insights

HYPEX discovers parameter-rules to gain knowledge on (a) the relationship between parameters and data characteristics, and (b) between hyperparameters themselves. We summarize identified parameter relationships for the six evaluated anomaly detectors in Table 5.1. It shows parameter dependencies per algorithm and the base oscillations for which we discovered the respective relation. Note that the base oscillations *random walk* and *cylinder bell funnel* are acyclical, which is why they cannot contain time series mutations in the frequency attribute. Our evaluation shows that the number and type of dependencies found by HYPEX for an arbitrary but fixed anomaly detector highly vary between different base oscillation behaviors. However, for all anomaly detectors except DWT-MLEAD, HYPEX identifies a data-dependent parameters corresponding to internal window sizes. These window size parameters depend on the base oscillation's frequency, its variance, the anomalies' length, or the dataset contamination, which is the ratio of anomalous to the total number of data points in the time series. Furthermore, while most hyperparameters purely depend on data characteristics, we also identified three dependencies between two different hyperparameters, namely *query window size* $\rightarrow$ *window size* for Series2Graph on base oscillations ECG and random walk, *n neighbors* $\rightarrow$ *window size* for Sub-LOF on base oscillation ECG, and *quantile epsilon* $\rightarrow$ *start level* for DWT-MLEAD on base oscillation sine.

Algorithm	Parameter	Dependency	Base Oscillations			
			Sine	ECG	RW †	CBF †
Donut [48]	window size	base-oscillation.frequency	+	+		
		base-oscillation.variance	+	+	+	+
		anomalies.length	+			
		contamination	+			
	epochs	anomalies.length				+
STOMP [51]	anomaly window size	anomalies.length	+	+	+	
		base-oscillation.variance				+
		contamination			+	
Series2Graph [7]	query window size	anomalies.length		+		
		base-oscillation.variance	+	+		
		contamination	+	+		
	window size	base-oscillation.variance		+	+	
		▲ query window size		+	+	
	rate	anomalies.length			+	
		base-oscillation.variance			+	+
		contamination	+		+	
	random state	contamination	+			
Sub-IF [27]	n trees	anomalies.length				+
		base-oscillation.variance		+		
		contamination			+	
	window size	base-oscillation.variance	+			+
DWT-MLEAD [42]	start level	contamination		+		
		▲ quantile epsilon	+			
Sub-LOF [9]	window size	anomalies.length			+	
		base-oscillation.variance		+	+	+
		contamination	+			
		▲ n neighbors		+		
	n neighbors	anomalies.length		+		
		base-oscillation.frequency		+		

Table 5.1: Parameter dependencies found for algorithms across the four base oscillations (i) sine, (ii) electrocardiogram (ECG), (iii) Random Walk (RW), and (iv) Cylinder Bell Funnel (CBF).
† Non-cyclic base oscillation, thus the data characteristic *base-oscillation.frequency* is not available.
▲ Dependency on another detector hyperparameter.

6 Conclusion

In this thesis, we addressed the time consuming process of tuning hyperparameters and proposed a system to extract parameter rules that can be used to transfer knowledge on (a) dependencies between parameters and data characteristics, and (b) between two parameters onto yet unseen application data. While previous work included manually crafted heuristics [35] or long-running optimization tasks [31, 36, 12, 24] which both require labels on large test datasets, our work provides an automated approach using synthetic datasets to derive parameter calculation rules based on identified causal relationships. In our evaluation, HYPEX's parameter suggestions outperformed the anomaly detectors' default parameters as well as other authors' heuristics across different anomaly detection methods and base oscillations. We showed that optimized fixed parameters perform well on a variety of different datasets at the cost of higher variance. HYPEX's approach using a mixture of parameter-rules and fixed parameter values in combination with the automatic parameter model selection, however, predicts well-performing hyperparameters on the vast majority of evaluation datasets.

Future Work This thesis focuses on numerical data characteristics and parameters only. Though, many systems expose hyperparameters of categorical data type as well. Thus, extending the parameter-rule-discovery to identify causal relationships between (a) two categorical parameters, or (b) a single categorical and numerical parameter might prove to provide even better performance scores. Furthermore, future work includes the application and evaluation of HYPEX in other domains such as data cleaning, machine learning, or pattern mining.

Bibliography

[1] Subutai Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha. "Unsupervised real-time anomaly detection for streaming data". In: *Neurocomputing* (2017).

[2] Mohiuddin Ahmed, Abdun Naser Mahmood, and Jiankun Hu. "Outlier detection". In: *The state of the art in intrusion prevention and detection* (2014).

[3] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. "Optuna: A next-generation hyperparameter optimization framework". In: *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (SIGKDD)*. 2019.

[4] E. Aleskerov, B. Freisleben, and B. Rao. "CARDWATCH: a neural network based database mining system for credit card fraud detection". In: *Proceedings of the IEEE/IAFE Computational Intelligence for Financial Engineering (CIFEr)*. 1997.

[5] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. "Algorithms for hyper-parameter optimization". In: *Advances in Neural Information Processing Systems (NIPS)* (2011).

[6] James Bergstra and Yoshua Bengio. "Random search for hyper-parameter optimization." In: *Journal of Machine Learning Research (JMLR)* (2012).

[7] Paul Boniol and Themis Palpanas. "Series2Graph: Graph-Based Subsequence Anomaly Detection for Time Series". In: *Proceedings of the VLDB Endowment (PVLDB)* (2020).

[8] Andrew P Bradley. "The use of the area under the ROC curve in the evaluation of machine learning algorithms". In: *Pattern Recognition* (1997).

[9] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. "LOF: identifying density-based local outliers". In: *Proceedings of the International Conference on Management of Data (SIGMOD)*. 2000.

[10] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly Detection: A Survey". In: *Journal of the ACM* (2009).

[11] Qing Chen, Anguo Zhang, Tingwen Huang, Qianping He, and Yongduan Song. "Imbalanced dataset-based echo state networks for anomaly detection". In: *Neural Computing and Applications* (2020).

[12] Jessamyn Dahmen and Diane J Cook. "Indirectly Supervised Anomaly Detection of Clinically Meaningful Health Events from Smart Home Data". In: *ACM Transactions on Intelligent Systems and Technology (TIST)* (2021).

[13] Jesse Davis and Mark Goadrich. "The relationship between Precision-Recall and ROC curves". In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2006.

[14] Ian Dewancker, Michael McCourt, and Scott Clark. *Bayesian Optimization for Machine Learning : A Practical Guidebook*. 2016.

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

[16] Ryohei Fujimaki, Takehisa Yairi, and Kazuo Machida. "An Approach to Spacecraft Anomaly Detection Problem Using Kernel Feature Space". In: *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (SIGKDD)*. 2005.

[17] Clark Glymour, Kun Zhang, and Peter Spirtes. "Review of causal discovery methods based on graphical models". In: *Frontiers in Genetics* (2019).

[18] Nico Görnitz, Marius Kloft, Konrad Rieck, and Ulf Brefeld. "Toward supervised anomaly detection". In: *Journal of Artificial Intelligence Research (JAIR)* (2013).

[19] Niklas Heim and James E Avery. "Adaptive anomaly detection in chaotic time series with a spatially aware echo state network". In: (2019).

[20] Geoffrey E Hinton. "A practical guide to training restricted Boltzmann machines". In: *Neural networks: Tricks of the trade*. 2012.

[21] Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. "An Efficient Approach for Assessing Hyperparameter Importance". In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2014.

[22] Chunggyeom Kim, Jinhyuk Lee, Raehyun Kim, Youngbin Park, and Jaewoo Kang. "DeepNAP: Deep neural anomaly pre-detection in a semiconductor fab". In: *Information Sciences* (2018).

[23] V. Kumar. "Parallel and distributed computing for cybersecurity". In: *IEEE Distributed Systems Online* (2005).

[24] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. "Efficient backprop". In: *Neural networks: Tricks of the trade*. 2012.

[25] Zeyan Li, Wenxiao Chen, and Dan Pei. "Robust and unsupervised kpi anomaly detection based on conditional variational autoencoder". In: *IEEE International Performance Computing and Communications Conference (IPCCC)*. 2018.

[26] Dapeng Liu, Youjian Zhao, Haowen Xu, Yongqian Sun, Dan Pei, Jiao Luo, Xiaowei Jing, and Mei Feng. "Opprentice: Towards practical and automatic anomaly detection through machine learning". In: *Proceedings of the Internet Measurement Conference (IMC)*. 2015.

[27] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. "Isolation forest". In: *IEEE International Conference on Data Mining (ICDM)*. 2008.

[28] C-T Lu, Dechang Chen, and Yufeng Kou. "Algorithms for spatial outlier detection". In: *IEEE International Conference on Data Mining (ICDM)*. 2003.

[29] Mohsin Munir, Shoaib Ahmed Siddiqui, Andreas Dengel, and Sheraz Ahmed. "DeepAnT: A deep learning approach for unsupervised anomaly detection in time series". In: *IEEE Access* (2018).

[30] Caleb C Noble and Diane J Cook. "Graph-based anomaly detection". In: *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (SIGKDD)*. 2003.

[31] Martin Pelikan, David E Goldberg, Erick Cantú-Paz, et al. "BOA: The Bayesian optimization algorithm". In: *Proceedings the Genetic and Evolutionary Computation Conference (GECCO)*. 1999.

[32] Jonas Peters, Joris M Mooij, Dominik Janzing, and Bernhard Schölkopf. "Causal discovery with continuous additive noise models". In: *Journal of Machine Learning Research (JMLR)* (2014).

[33] Matthew Rocklin. "Dask: Parallel computation with blocked algorithms and task scheduling". In: *Proceedings of the Python in Science Conference (SciPy)*. 2015.

[34] Stan Salvador, Philip Chan, and John Brodie. "Learning States and Rules for Time Series Anomaly Detection." In: *The Florida Artificial Intelligence Research Society Conference (FLAIRS)*. 2004.

[35] Sebastian Schmidl, Phillip Wenig, and Thorsten Papenbrock. "Anomaly Detection in Time Series: A Comprehensive Evaluation". In: *Proceedings of the VLDB Endowment* (2022).

[36] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. "Taking the human out of the loop: A review of Bayesian optimization". In: *Proceedings of the IEEE* (2015).

[37] Shashi Shekhar, Chang-Tien Lu, and Pusheng Zhang. "Detecting graph-based spatial outliers: algorithms and applications (a summary of results)". In: *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (SIGKDD)*. 2001.

[38] C. Spence, L. Parra, and P. Sajda. "Detection, synthesis and compression in mammographic image analysis with a hierarchical image probability model". In: *Proceedings IEEE Workshop on Mathematical Methods in Biomedical Image Analysis (MMBIA)*. 2001.

[39] Peter Spirtes, Clark N Glymour, Richard Scheines, and David Heckerman. *Causation, prediction, and search*. MIT press, 2000.

[40] Jiankai Sun, Deepak Ajwani, Patrick K. Nicholson, Alessandra Sala, and Srinivasan Parthasarathy. "Breaking Cycles In Noisy Hierarchies". In: *Proceedings of the ACM Web Science Conference (WebSci)*. 2017.

[41] Pei Sun, Sanjay Chawla, and Bavani Arunasalam. "Mining for outliers in sequential databases". In: *Proceedings of the SIAM International Conference on Data Mining (SDM)*. 2006.

[42] Markus Thill, Wolfgang Konen, and Thomas Bäck. "Time series anomaly detection with discrete wavelet transforms and maximum likelihood estimation". In: *International Conference on Time Series (ITISE)*. 2017.

[43] Yi Wang, Linsheng Han, Wei Liu, Shujia Yang, and Yanbo Gao. "Study on wavelet neural network based anomaly detection in ocean observing data series". In: *Ocean Engineering* (2019).

[44] Christina Warrender, Stephanie Forrest, and Barak Pearlmutter. "Detecting intrusions using system calls: Alternative data models". In: *Proceedings of the IEEE Symposium on Security and Privacy (SP)*. 1999.

[45] Andreas S Weigend, Morgan Mangeas, and Ashok N Srivastava. "Nonlinear gated experts for time series: Discovering regimes and avoiding overfitting". In: *International Journal of Neural Systems* (1995).

[46] Phillip Wenig and Sebastian Schmidl. *GutenTAG - A good Timeseries Anomaly Generator*. `https://github.com/HPI-Information-Systems/gutentag`. 2022.

[47] Phillip Wenig, Sebastian Schmidl, and Thorsten Papenbrock. "TimeEval: A Benchmarking Toolkit for Time Series Anomaly Detection Algorithms". In: *Proceedings of the VLDB Endowment* (2022).

[48] Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei, Yang Feng, et al. "Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications". In: *Proceedings of the International World Wide Web Conference (WWW)*. 2018.

[49] Torsten Zesch and Iryna Gurevych. "Analysis of the Wikipedia category graph for NLP applications". In: *Proceedings of the Second Workshop on TextGraphs: Graph-Based Algorithms for Natural Language Processing*. 2007.

[50] Qinyi Zhang, Sarah Filippi, Seth Flaxman, and Dino Sejdinovic. "Feature-to-feature regression for a two-step conditional independence test". In: *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)* ().

[51] Yan Zhu, Zachary Zimmerman, Nader Shakibay Senobari, Chin-Chia Michael Yeh, Gareth Funning, Abdullah Mueen, Philip Brisk, and Eamonn Keogh. "Matrix Profile II: Exploiting a Novel Algorithm and GPUs to Break the One Hundred Million Barrier for Time Series Motifs and Joins". In: *IEEE International Conference on Data Mining (ICDM)*. 2016.

Eidesstattliche Erklärung

Hiermit versichere ich, dass meine Masterarbeit "HYPEX: Explainable Hyperparameter Optimization in Time Series Anomaly Detection" ("HYPEX: Interpretierbare Parameter Optimierung für Anomalie Erkennung auf Zeitserien") selbständig verfasst wurde und dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt wurden. Diese Aussage trifft auch für alle Implementierungen und Dokumentationen im Rahmen dieses Projektes zu.

Potsdam, den September 14, 2022,

(Mats Pörschke)