HYPEX: Hyperparameter Optimization in Time Series Anomaly Detection

Sebastian Schmidl[1], Phillip Wenig[2], Thorsten Papenbrock[3]

Abstract:

Anomaly detection is a popular activity in time series analytics and covers various techniques for the identification of rare data patterns. These techniques are often presented in the form of automatic anomaly detection algorithms, whose performance depends significantly on the configuration of hyperparameters. Frequently, specifying the hyperparameters of an anomaly detection algorithm manually is particularly difficult because it requires an in-depth understanding of the data and the algorithms' internal behavior. While automatic methods for hyperparameter optimization exist, they require labeled training data and many trials to assess a system's performance before the system can be applied to production data. Hence, existing methods basically shift the efforts from parameter optimization to the labelling of datasets, which is – due to a general lack of high-quality, domain-specific labeled training data – a complex and time-consuming task in time series analytics.
In this paper, we propose a novel hyperparameter optimization framework called HYPEX that learns a parameterization model for anomaly detection algorithms from synthetically generated training data. Based on a (user provided or automatically measured) description of a few time series characteristics, HYPEX quickly suggests effective settings for unseen datasets. The suggestions are based on (i) explainable hyperparameter rules and (ii) learned default parameters, and require no labels for the to-be-analyzed target time series. Our evaluation shows that HYPEX' suggestions significantly improve an algorithm's performance compared to the algorithms' default values and handcrafted heuristics; they often even compete well with the optimal performance achieved with full Bayesian optimization.

Keywords: Time Series Anomaly Detection; Bayesian Optimization; Causal Discovery

1 The Curse of Hyperparameters

Anomaly detection algorithms for time series data analyze sequences of real-valued, usually time-dependent data for rare subsequence patterns, called anomalies. To obtain good results with these algorithms, various hyperparameters need to be specified. Many of these hyperparameters are algorithm-specific and cover properties, such as *learning rates*, *window sizes*, *maximum cardinalities*, *move distances*, *node degrees*, and *neighbor counts*, some of which hidden behind cryptic names, such as *k*, *phi*, or *delta*. What makes the specification of these hyperparameters hard is that (i) their implications are often hard to guess even by technical experts, (ii) the algorithmic performance is often highly sensitive to the chosen

[1] Hasso Plattner Institute, University of Potsdam, Germany, sebastian.schmidl@hpi.de
[2] Hasso Plattner Institute, University of Potsdam, Germany, phillip.wenig@hpi.de
[3] Philipps-Universität Marburg, Germany, papenbrock@informatik.uni-marburg.de

doi:10.18420/BTW2023-22

settings, (iii) most hyperparameters are numeric with an infinite parameter space, and (iv) the optimal values often depend on certain characteristics of the input data. For example, many time series anomaly detection algorithms take a *window size* as input. The optimal value for this hyperparameter then correlates, i. a., with the time series' base oscillation frequency, the expected anomaly length, or the time series' extreme values. Most anomaly detection algorithms, therefore, do not perform well with their default parameterization [SWP22].

Hyperparameter optimization is the process of tuning the hyperparameters of an algorithm to a well performing setting [FH19]. This process can be conducted manually or automatically. While the manual search is largely based on domain knowledge, automated approaches find optimal values via, e. g., systematic *Bayesian Optimization* [DC21; DMC16; PGC+99; Sh15] or comprehensive *Grid Search* [Hi12; Le12]. To make any of these approaches work, labeled training data is required to measure the quality of specific hyperparameter settings. Following established machine learning practice, we would collect possibly many labeled time series, and use classical hyperparameter optimization to find optimal settings for some anomaly detection algorithm. The *default values* found with this global optimization will likely perform poorly on a given target dataset because they cannot consider that some hyperparameter values, which are often the most important ones w. r. t. detection accuracy, depend on time series characteristics. Furthermore, real-world data is regularly poorly labeled and, hence, hardly usable for machine learning [SWP22]. For this reason, we would require high-quality training data for every input dataset to effectively optimize an anomaly detection algorithm. However, most practical use cases for time series anomaly detection do not offer any labeled training data, and labeling a sufficient amount of training data is hard.

To ease the hyperparameter optimization process and improve upon globally defined default values, we propose a novel approach: Given a to-be-analyzed target dataset and a to-be-optimized anomaly detection algorithm, we ask the user to specify a set of dataset characteristics, such as *variance*, *min* and *max* values, *oscillation frequencies*, and *expected anomaly lengths*. These characteristics can mostly be profiled automatically from the input data; if profiling is not possible, they are still easier to guess than hyperparameter values and easier to provide than a sufficient amount of labeled training data. Our novel system, then, generates training data based on the provided data characteristics and optimizes the algorithm on this data. The optimization of the hyperparameters is an effective but also expensive process. It, therefore, yields not only a set of possibly robust default parameters, but also a generalizable parameterization model. Because the model learned the relationship between data characteristics and optimal hyperparameter settings, it can quickly optimize the algorithm for different target datasets with different characteristics.

More specifically, we propose *Hyper Parameter Explanation* (HYPEX), a hyperparameter optimization framework that provides hyperparameter optimization models consisting of explainable parameter rules. Assuming that data scientists can specify *discrete* hyperparameters, such as preprocessors, CPU/GPU switches, or ML model types, easily (or at least intuitively), HYPEX focuses on the optimization of *continuous*, numerical hyperparameters. For this, HYPEX uses synthetically generated datasets to determine optimal hyperparam-

Algorithm	Family [SWP22]	Float	Integer	Total
STOMP [Zh16]	distance	1	2	3
DWT-MLEAD [TKB17]	distribution	1	2	3
Series2Graph [BP20]	encoding	-	4	4
Sub-LOF [Br00]	distance	-	5	5
Donut [Xu18]	reconstruction	1	5	6
Sub-IF [LTZ08]	trees	2	3	5

Tab. 1: Selection of anomaly detection approaches and their hyperparameters by category.

eter values and to identify relationships between (a) different hyperparameters and (b) hyperparameters and dataset characteristics. To learn a reliable hyperparameter model, HYPEX takes as input (i) a time series anomaly detection *algorithm*, (ii) a *hyperparameter configuration* that defines the to-be-optimized algorithm parameters, and (iii) a *dataset generation configuration* that describes possible dataset characteristics. Both configurations require the user to specify value ranges for the hyperparameters and data characteristics, respectively. These ranges should cover the expected use cases and define the scope of the optimization – the larger the ranges, the more general the trained model and the longer the training time. The dataset generation configuration, for example, might specify an oscillation frequency between 1Hz and 2Hz, a maximum value between 0.8 and 1.0, and anomaly lengths between 5s and 30s. Any target dataset in this scope can later be parameterized. Because HYPEX optimizes only numerical hyperparameters, the configurations need to provide settings for all discrete hyperparameters. Once started, HYPEX uses the configurations for automatic training data generation with the GutenTAG [WSP22] dataset generator and for specifying the trials in a systematic *Bayesian Optimization* [PGC+99]. From the many optimal configurations on different datasets, the system then distills all dependencies between hyperparameters and data characteristics into a causal parameter model. With the learned model, we can use HYPEX on any dataset with characteristics in the before specified ranges to propose optimal settings. For this, the user provides the (profiled or estimated) dataset characteristics of a concrete input dataset such that HYPEX can apply them to the parameter rules. Note that all discrete settings, such as type of input data or learning strategy, need to match the settings of the training. With the learned default values and parameter dependencies, HYPEX finally derives well-performing hyperparameter values.

We evaluate HYPEX on the six algorithms shown in Tab. 1 using synthetic datasets in Sect. 4.2. In Sect. 4.4, we exemplarily solve the hyperparameter optimization task for the algorithm Sub-LOF [Br00] on five real-world time series. The algorithms are well-performing representatives from five anomaly detection families that we introduced in a larger evaluative study [SWP22]; they contain between three and six hard-to-optimize numerical hyperparameters of type *float* or *integer*. Our evaluation shows that the automatically suggested hyperparameters improve the anomaly detection quality of the algorithms significantly compared to their default parameters and a manual, heuristics-driven parameterization approach from related work [SWP22]. In summary, HYPEX makes the following contributions:

(i) **Training data generation:** We extend the GutenTAG data generator with mutation rules to explore numeric dataset characteristics (Sect. 3.1).

(ii) **Workload distribution:** We design a distributed system for the scalable execution of very many Bayesian optimization tasks (Sect. 3.2).

(iii) **Causal structure learning:** We propose an algorithm for the identification of dependencies between numeric dataset characteristics and hyperparameters (Sect. 3.3).

(iv) **Parameter inference:** We discuss the automatic inference of hyperparameters from Baysian optimization runs on generated data (Sect. 3.4).

2 Related Work

Optimizing hyperparameters is a well-known task in many research areas. The three most common approaches for this task are (i) *Random Search* [BB12], (ii) *Grid Search* [Hi12; Le12], and (iii) *Bayesian Optimization* [DMC16; PGC+99; Sh15]. 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. All three algorithms require large amounts of labelled data to evaluate the algorithm or machine learning (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* [Li15] is an interesting approach that uses supervised machine learning to improve the quality of anomaly detection in practice. The proposed algorithm 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. Moreover, domain experts are required to label anomalies in the incoming real-world data. 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 *Isudra* [DC21] was developed in the context of detecting anomalous data points in clinical health data. The indirect supervision approach for anomaly detection methods

serves to optimize existing unsupervised anomaly detectors and to tune them to concrete application settings. The approach requires clinicians to label sensor time series data with 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 the alternative methods iForest and One-Class SVM in detecting six out of seven health events. 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 requirement of domain experts providing anomaly labels by using a data generator that generates synthetic ground truth data with anomaly labels. Moreover, our work returns a set of parameter-rules, which can be used to adjust a detector's parameters to new, yet unseen datasets.

3 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 relationships between hyperparameters themselves. Because we demonstrate HYPEX in the domain of time series anomaly detection, we first introduce the data (time series) and algorithms (anomaly detection algorithms), we work with.

A *time series* is an ordered sequence $T = \{T_0, T_1, ..., T_{n-1}, T_n\}$ of real-valued data points $T_i \in \mathcal{R}^m$. An anomaly in such a time series is a subsequence of data points that deviates w. r. t. some measure or model from the frequent patterns in the time series. In our work, we consider w.l.o.g. only univariate time series with a single attribute per data point ($m = 1$). An anomaly detector takes a time series T as input and computes an anomaly score $s_i \in \{0, 1\}$ for each data point $T_i \in T$. The anomaly score s indicates the detector's confidence that a data instance T_i is anomalous. The anomaly scores cannot be compared to the anomaly labels unless they are turned into binary labels using a threshold. Hence, the choice of threshold significantly impacts the anomaly detector's performance and to eliminate it as another tuning parameter, we use the *AUC-PR* score [Br97; DG06] as a performance measure for the anomaly detectors' outputs. This measure is especially popular in applications dealing with learning on imbalanced data, which is the case for anomaly detection.

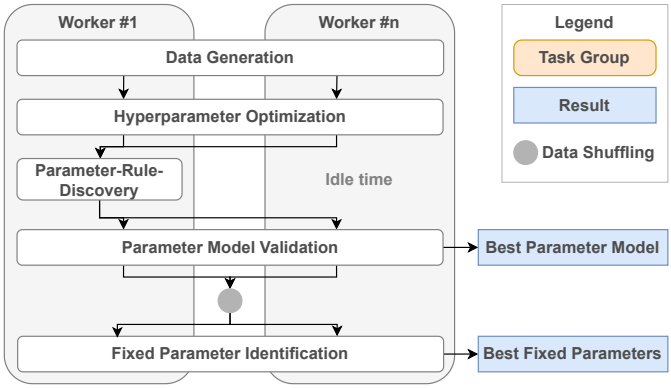


Fig. 1: Control flow of a distributed HYPEX execution consisting of five major steps: (i) data generation (Sect. 3.1), (ii) hyperparameter optimization (Sect. 3.2), (iii) parameter-rule-discovery (Sect. 3.3), (iv) parameter model validation (Sect. 3.3.5), and (v) fixed parameter identification (Sect. 3.4).

HYPEX consists of five consecutive steps that are explained in the following sections: (i) The *data generation* step creates time series, injects anomalies, and labels them (Sect. 3.1), (ii) the *hyperparameter optimization* step uses Bayesian Optimization to find optimal parameters on the generated datasets (Sect. 3.2), (iii) the *parameter-rule-discovery* step finds causal dependencies between data characteristics and the detector's hyperparameters, which creates a set of parameter model candidates (Sect. 3.3.1 to 3.3.4), (iv) the *parameter model selection* step validates the parameter model candidates to produce the final parameter model that incorporates all significant parameter rules (Sect. 3.3.5), and (v) the *fixed parameter identification* step finds data-independent, fixed parameter values for those parameters that are not covered by the parameter model (Sect. 3.4). The resulting final parameter model and the fixed parameters can be used to calculate future hyperparameters on yet unseen time series based on that time series' characteristics. It is worth noting that training the parameter model, including data generation, Baysian optimization and causal inference, is a costly process. For this reason, we propose to parallelize and distribute the efforts. But the training enables HYPEX to afterwards parameterize an algorithm for different input datasets in constant time. Fig. 1 provides a high-level architecture overview of our HYPEX approach. All time-intensive tasks, namely steps (i), (ii), (iv) and (v), are executed on a distributed Dask [Ro15] cluster utilizing multiple parallel worker nodes to speed up the overall runtime.

3.1 Data Generation

We use the time series anomaly data generator GutenTAG [WSP22] to generate the synthetic datasets used to optimize the anomaly detector's parameters. Each generated time series has a base oscillation behavior and potentially multiple injected anomalies of different types. GutenTAG pre-defines six base oscillations and nine anomaly types. When generating a

time series datasets, GutenTAG provides access to the time series, the anomaly labels, and the generation metadata. This information is used to extract and control the dataset characteristics. Fig. 2 shows a generated, univariate, real-valued time series, where the underlying base oscillation simulates electrocardiogram (ECG) data. Two anomalies were added, both indicated by the red background color: the first at position 140 of type *frequency* and length 12, the second at position 240 of type *extremum* and length 1.

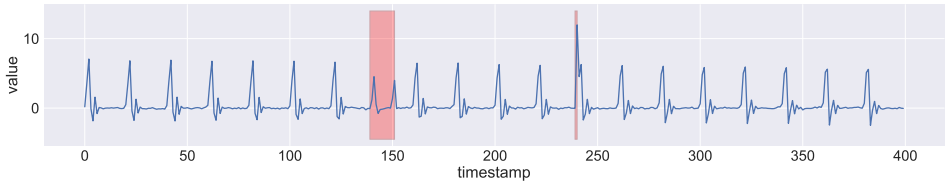


Fig. 2: Exemplary time series generated with GutenTag [WSP22]

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 multiple time series mutations, so-called *mutation sets*, which means that two datasets can differ in multiple characteristics. HYPEX uses user-defined mutation sets to generate synthetic datasets that vary in an arbitrary but fixed set of data characteristics, such as anomaly lengths, oscillation frequencies, variances etc. Fig. 3 shows an example of a mutation set that contains a single time series mutation of the attribute *base-oscillation.frequency* applied to the 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 `n_samples` = 50 mutated GutenTAG configurations. Subsequently, those 50 different GutenTAG configurations are used to generate the actual 50 datasets. More datasets improve the training quality, but also increase the training time; our experiments use a generously high value of 50 because we focus on quality. By randomly selecting dataset characteristics from the specified ranges, the generated data has all the important characteristics of real data, but with a certain variety and controlled anomalies – for this reason, the learnings on the generated data translate well to real data. To conclude the data generation, HYPEX splits the set of generated time series datasets 60:20:20 into distinct train, validation, and test sets. All physical nodes in the Dask cluster perform the data generation task concurrently.

3.2 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. Each step, i. e., a particular hyperparameter configuration, of the optimization process is called trial and HYPEX keeps track of the dataset,

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

Fig. 3: Exemplary mutation of the data characteristic *base-oscillation.frequency* of a user-defined time series called *ecg-data*.

its characteristics, the tested hyperparameters, and the resulting performance scores for all performed trials over all datasets. The number of required trials to achieve reliable results strongly depends on the number and data types of the hyperparameters to optimize.

Optimization Procedure HYPEX uses *Bayesian Optimization* for the optimization procedure. While *Random Search* and *Grid Search* simply return the best performing hyperparameters after testing several parameter configurations, *Bayesian Optimization* bases hyperparameter guesses on past parameter evaluations, leading to a faster convergence. Bayesian optimization is commonly used to optimize an objective function f [Fr18] that is expensive to evaluate, such as tuning an ML model's architecture, or finding the best hyperparameters for an anomaly detection algorithm. 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 [DMC16]. Until a time constraint C is met, SMBO keeps repeating the following four steps: (i) update the probabilistic model based on previously collected benchmark results, (ii) select the next best guess parameters $\mathcal{H}$ based on the probabilistic model, (iii) evaluate the objective function f with the parameters $\mathcal{H}$, which is the most expensive step, and (iv) collect and save the benchmark results $(\mathcal{H}, f(\mathcal{H}))$ for the upcoming optimization steps. The research community came up with several samplers to generate the next best parameter guess based on the previous evaluated parameters [Be11]. 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 [Ak19], which implements the TPE algorithm with its *TPESampler*, and we use the *AUC-PR* score [Br97; DG06] as the optimization criterion.

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 [Ro15]. 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.

3.3 Parameter Rule Discovery

We use the optimized hyperparameters found by Optuna (Sect. 3.2) to discover *parameter rules* that specify (a) how hyperparameters depend on other hyperparameters and (b) how hyperparameters depend on specific data characteristics. These data characteristics are represented by the applied time series mutations in our optimization process. To discover the parameter rules, we first present a de-noising step for the trial results, which is the basis of HYPEX's rule discovery (Sect. 3.3.1). Then, we guide through the estimation of an undirected causal graph, the so-called causal skeleton (Sect. 3.3.2). Because the causal skeleton is undirected, we then need to orientate the edges into determinant and dependent hyperparameters/characteristics (Sect. 3.3.3). Once the dependence graph is completed, 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 (Sect. 3.3.4). Finally, we present an approach to validate discovered parameter rules (Sect. 3.3.5).

3.3.1 Noise Reduction

The set of collected Optuna trials consists of hyperparameter configurations with different performance AUC-PR scores. To find parameter rules, HYPEX should, however, consider only such trails that performed well, because configurations that led to poor trail results

with low AUC-PR scores are not indicative for hyperparameter/characteristic correlations and must, therefore, be considered 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 threshold γ as the top 10% AUC-PR score quantile, which has shown to be a robust selection strategy in all our experiments.

To further improve the explainability of the discovered parameter rules, we also prune parameter rules for hyperparameters with low impact on an anomaly detector's performance. For this, HYPEX uses the parameter importance evaluator fANOVA [HHL14] to identify important hyperparameters. 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 hyperparameters. Then, we prune all hyperparameters with an importance of less than 1% – these parameters, which are usually runtime performance related parameters, do not require optimization.

After noise reduction, HYPEX combines the noise-reduced trials and the information on hyperparameter importance into a matrix M of dimensionality $C \times N$. N defines the number of trials left after reducing the noise and $C = p + d + 1$ defines the number p of anomaly detector hyperparameters with an importance score ≥ 0.01 plus the number d of applied time series mutations 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 99.5% and *random state* with an importance score of 0.5% as input. Moreover, let the datasets contain the single applied time series mutation *base-oscillation frequency*. The resulting matrix M would consist of three columns, namely *window size*, *base-oscillation frequency*, and the AUC-PR score; *random state* was removed due to its low importance score of 0.5%. The number of rows in M depends on the performed trials' AUC-PR score distribution. With the assumption of a uniform AUC-PR score distribution and 300 trial runs, we expect M to have $0.1 \cdot 300 = 30$ rows.

3.3.2 Skeleton Estimation

To identify causal dependencies between columns of the noise-reduced data matrix M (Sect. 3.3.1), HYPEX performs 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 tiny signal-to-noise ratio. This is why we use a fairly simple, still powerful, regression-based non-linear independence test with the PC algorithm [Sp00]. The PC algorithm is one of the oldest methods to discover causal dependency graphs [GZS19; Sp00]. 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[4] and extend it with own regression-based independence and CI tests

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

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., hyperparameter 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 Sect. 3.3.5).

(iii) For each of the remaining edges $(A, B) \in G$ and each set of nodes $Z = \{Z_1, ..., Z_n\}$ with $n \in \mathbb{N}^+$ that are all either connected to A or 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 $A \rightarrow B \rightarrow C$. 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 $A \rightarrow C$ 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 the least squares optimization on pre-transformed data and comes with two transformations by default: *linear* and *hyperbola*. The *linear* 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 set of transformations is easily extendable, but our experiments strongly suggested that (at least in the domain of anomaly detection on time series data) *linear* and *hyperbola* dependencies are most common and, hence, sufficient. To fit the non-linear regression with these transformations, our approach takes the following inputs: 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 $\mathcal{T}$. For each of the provided transformation functions in $\mathcal{T}$, the algorithm transforms the input data of the predicting variable $\mathcal{X}$ and fits a linear regression on the transformed data $\mathcal{X}_T$:

$model \leftarrow fitLinearRegression(\mathcal{X}_T, \mathcal{Y}, \mathcal{W}^2)$. Then, 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. 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. Finally, the model with the highest R^2-Score and the R^2-Score itself are returned. To test for independence between $\mathcal{X}$ and $\mathcal{Y}$, HYPEX checks whether the calculated R^2-Score is smaller than a provided threshold α. Hence, the quality of the causal discovery output strongly depends on the chosen value for α. We provide more detailed information on how we use α to generate different model candidates in Sect. 3.3.5.

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. [Pe14] 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 of $\mathcal{Z}$. 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 $\mathcal{N}_x \perp \mathcal{N}_y$ [Zh17]. Algorithm 1 shows the application of this redefinition: The CI test uses two steps, which are (i) the estimation of the noise terms $\mathcal{N}_x$ and $\mathcal{N}_y$, and (ii) the test 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 of $\mathcal{Z}$ onto $\mathcal{X}$ (Line 2) and $\mathcal{Y}$ (Line 5); we then 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 (Line 8) to check whether the R^2-Score 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 Sect. 3.3.5).

Algorithm 1 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
```

3.3.3 Edge Orientation

Until now, we found correlated hyperparameter-hyperparameter and hyperparameter-characteristic edges as undirected relationships. To derive hyperparameters from other hyperparameters or dataset characteristics, these relationships need to be oriented, which is

done in three steps: (i) incorporation of prior knowledge on dataset characteristics, (ii) estimation of the Completed Partially Directed Acyclic Graph (CPDAG), and (iii) conversion of the CPDAG to a Directed Acyclic Graph (DAG).

In step (i), we incorporate a task-specific constraint: Because we aim to find good hyperparameters given certain dataset characteristics, edges need to point from characteristics to parameters. Hence, given the edge (A, B) with orientation $A \rightarrow B$, HYPEX removes all edges (A, B) from the graph skeleton where the node B is a dataset characteristic. Step (ii) then 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 [GZS19]. 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 [GZS19]:

(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 $A \rightarrow B$ and $C \rightarrow B$.

(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 $B \rightarrow C$.

The resulting CPDAG might still contain undirected edges, but our final parameter model requires all discovered parameter rules to have a clear orientation. To ensure this requirement, step (iii) of the edge orientation converts the estimated CPDAG to a DAG with a colored depth-first search [Su17; ZG07]. The search removes back edges in the graph, thus breaking existing cycles and creating a DAG with no undirected edges or cyclic dependencies.

3.3.4 Parameter Model

Given the dependency DAG, HYPEX now trains a set of *parameter models* that predict optimal values for dependent anomaly detector hyperparameters from the set of data characteristics and other hyperparameters: For each node in the DAG, 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 hyperparameter. Once the parameter models are trained on generated data, HYPEX can use them to predict hyperparameter values on unseen data. For this, the algorithm iterates over the estimated DAG in topological order. In each step, it uses the stored NLRegT model to predict the respective parameter based on all previously estimated parameters and data characteristics. In Sect. 3.4, we discuss how HYPEX assigns fixed values to the hyperparameters that are not covered by the parameter model.

3.3.5 Parameter Model Selection

Our experiments show that the optimal α and β thresholds to choose for the parameter-rule-discovery (Sect. 3.3.2) are 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 and considering only R^2-scores $\geq 5\%$ as significant; the set of β thresholds is fixed and set to $\{0.2, 0.4, 0.6, 0.8\}$. These settings have been found via systematic ablation tests and showed to be dataset and algorithm independent; hence, we propose them as default settings for HYPEX. 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, which contains 20% of the generated time series datasets. 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, finally, choose the parameter model with the highest mean AUC-PR score across all conducted tests.

3.4 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, 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 hyperparameter values of the best performing trial 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.

4 Evaluation

In this section, we evaluate HYPEX on a variety of anomaly detection algorithms and time series datasets. We start with the explanation of the experimental setup (Sect. 4.1), then compare the performance scores achieved by our parameter suggestions with relevant alternative approaches (Sect. 4.2), review HYPEX's sensitivity to automatically chosen thresholds (Sect. 4.3), and show the application of HYPEX to real-world data (Sect. 4.4).

4.1 Experimental Setup

We evaluate our approach[5] 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 [WSP22]. 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 and contain 3, 6, or 9 same-length anomalies at different positions of types (i) variance, (ii) frequency, or (iii) pattern [WSP22]. We apply time series mutations to (a) the base oscillation frequency (only applicable for sine and ECG), (b) the base oscillation variance, (c) the length of anomalies, and (d) the number of anomalies. 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 are reserved for evaluation purposes only (Sect. 3.1). We evaluate our approach on algorithms from five out of six anomaly detector families defined by Schmidl et al. [SWP22]. The algorithms stem from the areas (i) distance (STOMP [Zh16] and Sub-LOF [Br00]), (ii) distribution (DWT-MLEAD [TKB17]), (iii) encoding (Series2Graph [BP20]), (iv) reconstruction (Donut [Xu18]), and (v) trees (Sub-IF [LTZ08]). Each algorithm has between 3 and 6 hyperparameters that are subject to optimization. For the evaluation, we use the AUC-PR score to measure the algorithms' detection quality.

We do not have any information on the true relationships between data characteristics and hyperparameters for any of the anomaly detectors, on which we evaluate our framework HYPEX. Therefore, 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 of TimeEval [SWP22], and (c) the optimization results achieved by the Bayesian Optimizer Optuna *(full optimization)*. While each detector's default hyperparameter configuration stays constant, regardless of which time series it runs

[5] Code and evaluation scripts: `https://github.com/HPI-Information-Systems/hypex`

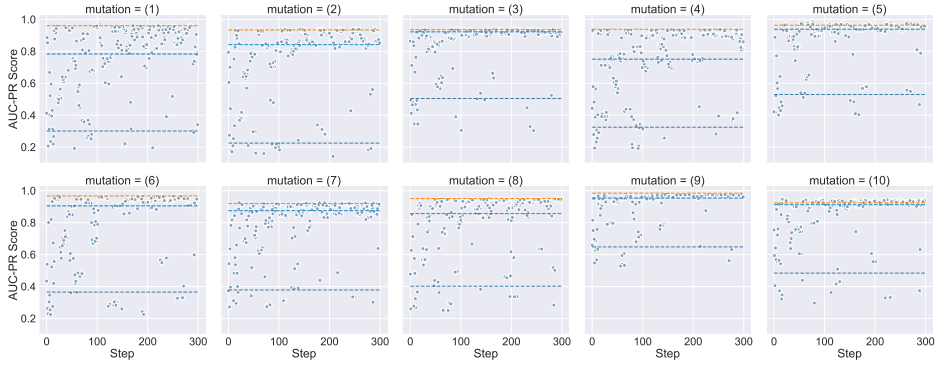


Fig. 4: Evaluation of Donut [Xu18] on 10 sine time series comparing our ·· *Parameter Model* with Donut's ·· *Default Parameters*, the ·· *Timeeval Heuristics*, and a • *Full Optimization* run.

on, HYPEX's parameter model as well as TimeEval's parameter suggestions 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, tunes every single parameter to the specific input time series. Thus, the best scores of the full optimization represent an upper bound for any optimization effort (which is achievable only with suitable training data) while the algorithm's default parameters represent a lower bound for HYPEX's performance; the TimeEval results show what results can be expected with significant manual effort.

4.2 Parameter Model Performance

In this section, we first 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.

Fig. 4 visualizes our evaluation results on the anomaly detector Donut [Xu18] and 10 time series datasets with base oscillation *sine* that cover various time series mutations with different dataset characteristics and, in particular, different types and numbers of anomalies in each time series. It compares the performance of ■ HYPEX's parameter model with the performance achieved by (i) ■ the detector's default parameters, (ii) ■ TimeEval's manually found default parameters and heuristics, and (iii) ■ a full Bayesian optimization run *(full optimization)*. While the default parameters, the TimeEval heuristics, and our parameter model recommend a single hyperparameter configuration per input dataset, the full optimization is granted 300 trials to optimize Donut's hyperparameters for each of the 10 time series. Our first insight is that especially Donut's default parameters perform poorly, which clearly emphasizes the need for hyperparameter tuning. In comparison to the algorithms' default values, both the TimeEval approach and HYPEX's parameter

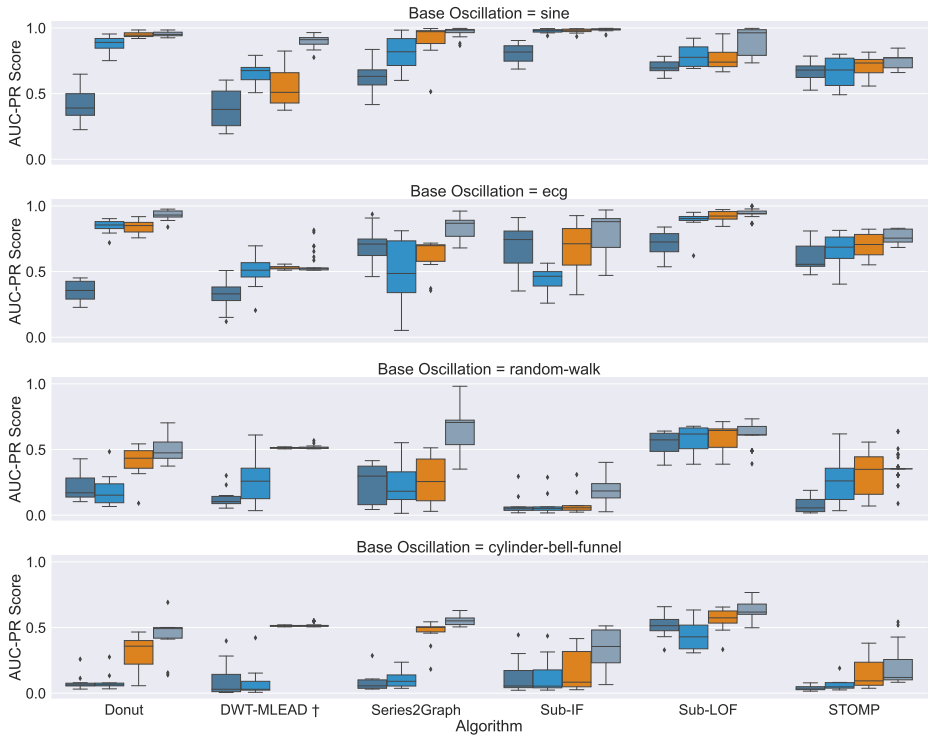


Fig. 5: Distribution of maximum AUC-PR scores achieved by *Default Parameters*, *Timeeval Heuristics*, our *Parameter Model*, and *Full Optimization* per base oscillation and algorithm.
† Empty parameter model for base oscillations *random walk* and *cylinder-bell-funnel*.

model achieve high AUC-PR scores; the difference is that no manual work was needed to find the parameter model. Furthermore, HYPEX's performance scores get even close to the maximum scores achieved via Bayesian optimization (consider the highest point for comparison); as an unsupervised parametrization approach, though, HYPEX does not require labels for the input time series to optimize it, but instead generates training data automatically.

We summarize our evaluation results across all six tested algorithms on four base oscillations in Fig. 5. A single box plot shows the distribution of 10 AUC-PR scores. Each score represents the evaluation result on a single evaluation time series dataset. To represent the Bayesian full optimization trials, we pick the maximum AUC-PR scores per time series dataset obtained in each of the 300 trial runs. As expected and confirmed in this experiment, a full optimization for a target dataset with training data performs best on all dataset types, i.e., base oscillations. Again, the algorithms' default parameters achieve the worst performance

scores in the majority of the conducted experiments. The manual hyperparameter settings and heuristics of TimeEval mostly deliver higher performance scores than the algorithms' default parameters. In some cases, such as Donut on random walk, Series2Graph on ECG and random walk, and Sub-IF on ECG, however, the TimeEval efforts could not predict better-performing hyperparameters than the default parameters. With no human effort and no pre-labeled training data, our parameter models' hyperparameter suggestions surpass both default parameterization and TimeEval performances in most experiments. Even in cases where HYPEX did not discover any parameter rules for an anomaly detector and simply suggested fixed values, these values still outperformed the detector's default parameters (see DWT-MLEAD on base oscillations sine, random walk, and cylinder bell funnel).

In terms of absolute performance, we see that all evaluated anomaly detectors tend to perform best on cyclical base oscillations, such as sine and ECG, and struggle with non-cyclical ones, such as random walk and cylinder bell funnel.

4.3 Sensitivity to Thresholds

HYPEX uses two thresholds, namely α and β, to adjust the minimum confidence scores of the (in-)dependence and CI tests. In Sect. 3.3.5, we discussed how the algorithm automatically determines the optimal values for α and β during the parameter model selection. Fig. 6 shows the parameter model results on the evaluation datasets when HYPEX performs the optimal threshold determination on the validation datasets. We optimized each parameter model's fixed parameters independently for each of the threshold tuples. Many experiments indicate that the full optimization trial runs have only 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. However, it 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. Hence, HYPEX automatically chooses

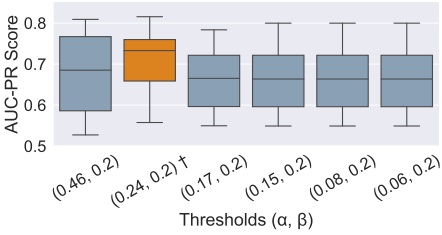
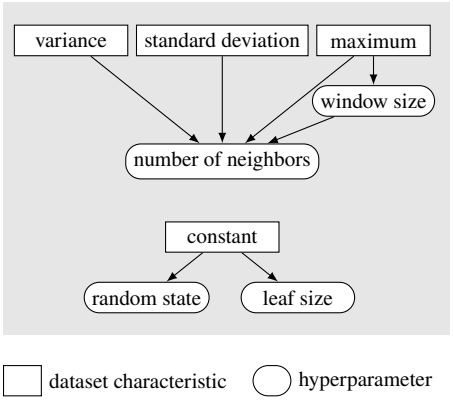


Fig. 6: Comparison of AUC-PR score distributions by selected α and β threshold values on 10 evaluation datasets for algorithm STOMP [Zh16] and base oscillation sine. Each named threshold tuple represents a set of tuples that all result in the identification of identical parameter-rules.
† The threshold value HYPEX automatically selected (see Sect. 3.3.5).

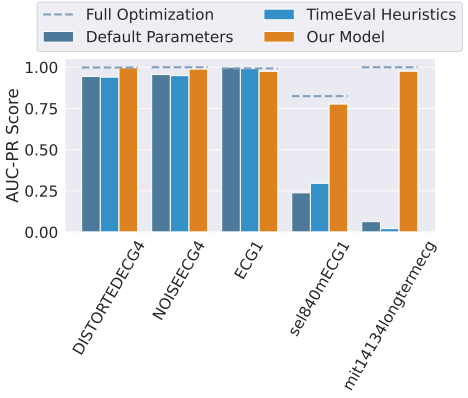
the threshold values $(0.24, 0.2)$. Across all experiments, the resulting parameter models contain a mixture of parameter rules and fixed hyperparameter values, which is a result that performs best on the evaluation datasets and shows to have a lower variance than using only fixed parameter values.

4.4 Application on real-world data

In this section, we evaluate how HYPEX' hyperparameter suggestions perform on five real-world datasets taken from the KDD-TSAD benchmark collection [Ke21]. All five time series represent ECG signals with varying properties, and contain different anomalies. Due to the space limitation and because Sub-LOF performed consistently well in all our previous evaluations, we restrict our experiment to the Sub-LOF algorithm. We configured HYPEX to optimize all four hyperparameters of Sub-LOF, which are `window size`, `number of neighbors`, `leaf size`, and `random state`, and to create datasets with similar data characteristics to the target datasets, i. e., ECG shaped data. We allow HYPEX to mutate `base oscillation frequency`, `noise level`, `amplitude`, `mean`, and anomaly details, such as `position`, `length`, and `anomaly shift size`. For both the generated training datasets and the real-world datasets, HYPEX uses *tsfresh* [Ch18] to extract 12 time series features as the dataset characteristics.



(a) HYPEX parameter model learned for the Sub-LOF algorithm on the synthetically generated training data.



(b) AUC-PR score of Sub-LOF on real datasets. HYPEX was trained only on synthetic data.

Fig. 7: HYPEX' parameter model and AUC-PR scores for Sub-LOF on real ECG datasets.

Fig. 7a shows HYPEX' final parameter model for Sub-LOF. The optimization on the synthetic training datasets identified dependencies for the hyperparameters `window size` and `number of neighbors`: While `window size` depends solely on the dataset characteristic `maximum`, `number of neighbors` dependents not only on dataset characteristics, but also on the hyperparameter `window size`. HYPEX assigned constant values to the parameters

`random state` and `leaf size` because they had no significant influence on the algorithm's performance on the training datasets.

In Fig. 7b, we show the AUC-PR scores of Sub-LOF on the five real-world datasets using (a) HYPEX' parameter model, (b) the algorithms' default parameters, (c) the parameter recommendations of TimeEval, and (d) the maximum of a full optimization run with 300 trials. The full optimization run indicates the optimal performance that Sub-LOF could achieve on each dataset when taking the ground truth into account. Both the default hyperparameters and the TimeEval hyperparameters perform poorly for the datasets *sel840mECG1* and *mit14134longtermecg*. For the dataset *ECG1*, the AUC-PR scores using the default values (1.00) and the hyperparameter values from TimeEval (1.00) are marginally higher than HYPEX' score (0.98). HYPEX' hyperparameter values, however, can achieve an AUC-PR score close to the full optimization run for all datasets. This demonstrates HYPEX's capability to learn a parameter model on synthetic training datasets that can significantly outperform alternative parametrization strategies, and that routinely approaches the maximum achievable score.

5 Conclusion

In this paper, we addressed the time-consuming process of tuning hyperparameters. Our proposed system HYPEX extracts parameter rules that can be used to transfer knowledge about the relationship (a) between parameters and data characteristics, and (b) between two parameters to yet unseen application data. While previous work included manually crafted heuristics [SWP22] or long-running optimization tasks [DC21; Le12; PGC+99; Sh15], which both require labels on large test datasets, our work proposes 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 hand-crafted heuristics across different anomaly detection methods and base oscillations. We showed that identified 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 hyperparameter values with the automatic parameter model selection, predicts well-performing, reliable hyperparameter values on different datasets. Future work includes the extension of HYPEX to categorical data types and its application and evaluation in other domains, such as data cleaning or pattern mining.

Acknowledgements
This paper is based on the excellent work of Mats Pörschke, whose career prevented him from officially co-authoring this publication. His master thesis[6] explains HYPEX in more detail. We thank Mats for his effort and permitting the presentation of the results.

[6] `https://github.com/HPI-Information-Systems/hypex/raw/main/masterthesis_hypex.pdf`

References

[Ak19] Akiba, T.; Sano, S.; Yanase, T.; Ohta, T.; Koyama, M.: Optuna: A next-generation hyperparameter optimization framework. In: Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (SIGKDD). 2019.

[BB12] Bergstra, J.; Bengio, Y.: Random search for hyper-parameter optimization. Journal of Machine Learning Research (JMLR) 13/1, 2012.

[Be11] Bergstra, J.; Bardenet, R.; Bengio, Y.; Kégl, B.: Algorithms for hyper-parameter optimization. In: Advances in Neural Information Processing Systems (NIPS). 2011.

[BP20] Boniol, P.; Palpanas, T.: Series2Graph: Graph-Based Subsequence Anomaly Detection for Time Series. Proceedings of the VLDB Endowment 13/12, 2020.

[Br00] Breunig, M. M.; Kriegel, H.-P.; Ng, R. T.; Sander, J.: LOF: identifying density-based local outliers. In: Proceedings of the International Conference on Management of Data (SIGMOD). 2000.

[Br97] Bradley, A. P.: The use of the area under the ROC curve in the evaluation of machine learning algorithms. Pattern Recognition 30/7, 1997.

[Ch18] Christ, M.; Braun, N.; Neuffer, J.; A.W., K.-L.: Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh – A Python package). Neurocomputing 307/1, 2018.

[DC21] Dahmen, J.; Cook, D. J.: Indirectly Supervised Anomaly Detection of Clinically Meaningful Health Events from Smart Home Data. ACM Transactions on Intelligent Systems and Technology (TIST) 12/2, 2021.

[DG06] Davis, J.; Goadrich, M.: The relationship between Precision-Recall and ROC curves. In: Proceedings of the International Conference on Machine Learning (ICML). 2006.

[DMC16] Dewancker, I.; McCourt, M.; Clark, S.: Bayesian Optimization for Machine Learning: A Practical Guidebook, 2016, arXiv: cs/1612.04858.

[FH19] Feurer, M.; Hutter, F.: Hyperparameter Optimization. In: Automatic Machine Learning: Methods, Systems, Challenges. Springer Berlin Heidelberg, 2019.

[Fr18] Frazier, P. I.: A Tutorial on Bayesian Optimization, 2018, arXiv: stat.ML/1807.02811.

[GZS19] Glymour, C.; Zhang, K.; Spirtes, P.: Review of causal discovery methods based on graphical models. Frontiers in Genetics 10/1, 2019.

[HHL14] Hutter, F.; Hoos, H.; Leyton-Brown, K.: An Efficient Approach for Assessing Hyperparameter Importance. In: Proceedings of the International Conference on Machine Learning (ICML). 2014.

[Hi12] Hinton, G. E.: A practical guide to training restricted Boltzmann machines. In: Neural networks: Tricks of the trade. Springer Berlin Heidelberg, 2012.

[Ke21] Keogh, E.; Dutta Roy, T.; Naik, U.; Agrawal, A.: Multi-dataset Time-Series Anomaly Detection Competition, 2021, URL: https://compete.hexagon-ml.com/practice/competition/39/, visited on: 11/09/2021.

[Le12] LeCun, Y. A.; Bottou, L.; Orr, G. B.; Müller, K.-R.: Efficient backprop. In: Neural networks: Tricks of the trade. Springer Berlin Heidelberg, 2012.

[Li15] Liu, D.; Zhao, Y.; Xu, H.; Sun, Y.; Pei, D.; Luo, J.; Jing, X.; Feng, M.: Opprentice: Towards practical and automatic anomaly detection through machine learning. In: Proceedings of the Internet Measurement Conference (IMC). 2015.

[LTZ08] Liu, F. T.; Ting, K. M.; Zhou, Z.-H.: Isolation forest. In: IEEE International Conference on Data Mining (ICDM). 2008.

[Pe14] Peters, J.; Mooij, J. M.; Janzing, D.; Schölkopf, B.: Causal discovery with continuous additive noise models. Journal of Machine Learning Research (JMLR) 15/1, 2014.

[PGC+99] Pelikan, M.; Goldberg, D. E.; Cantú-Paz, E., et al.: BOA: The Bayesian optimization algorithm. In: Proceedings the Genetic and Evolutionary Computation Conference (GECCO). 1999.

[Ro15] Rocklin, M.: Dask: Parallel computation with blocked algorithms and task scheduling. In: Proceedings of the Python in Science Conference (SciPy). 2015.

[Sh15] Shahriari, B.; Swersky, K.; Wang, Z.; Adams, R. P.; De Freitas, N.: Taking the human out of the loop: A review of Bayesian optimization. In: Proceedings of the IEEE. 2015.

[Sp00] Spirtes, P.; Glymour, C. N.; Scheines, R.; Heckerman, D.: Causation, prediction, and search. MIT press, 2000.

[Su17] Sun, J.; Ajwani, D.; Nicholson, P. K.; Sala, A.; Parthasarathy, S.: Breaking Cycles In Noisy Hierarchies. In: Proceedings of the ACM Web Science Conference (WebSci). 2017.

[SWP22] Schmidl, S.; Wenig, P.; Papenbrock, T.: Anomaly Detection in Time Series: A Comprehensive Evaluation. Proceedings of the VLDB Endowment 15/9, 2022.

[TKB17] Thill, M.; Konen, W.; Bäck, T.: Time series anomaly detection with discrete wavelet transforms and maximum likelihood estimation. In: International Conference on Time Series (ITISE). 2017.

[WSP22] Wenig, P.; Schmidl, S.; Papenbrock, T.: TimeEval: A Benchmarking Toolkit for Time Series Anomaly Detection Algorithms. Proceedings of the VLDB Endowment 15/12, 2022.

[Xu18] Xu, H.; Chen, W.; Zhao, N.; Li, Z.; Bu, J.; Li, Z.; Liu, Y.; Zhao, Y.; Pei, D.; Feng, Y., 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.

[ZG07] Zesch, T.; Gurevych, I.: 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.

[Zh16] Zhu, Y.; Zimmerman, Z.; Senobari, N. S.; Yeh, C.-C. M.; Funning, G.; Mueen, A.; Brisk, P.; Keogh, E.: 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.

[Zh17] Zhang, Q.; Filippi, S.; Flaxman, S.; Sejdinovic, D.: Feature-to-feature regression for a two-step conditional independence test. In: Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI). 2017.