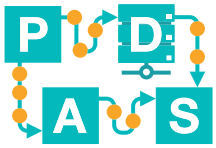
communicated by
Prof. Wil van der Aalst



RWTH AACHEN UNIVERSITY

The present work was submitted to:
Chair of Process and Data Science
RWTH Aachen University

Improving Process Discovery Through Hyperparameter Tuning

Bachelor Thesis

presented by
Georgi Georgiev
434925

First Examiner: Prof. Dr. Wil M. P. van der Aalst
Second Examiner: Prof. Dr. Sander Leemans

Advisor: Tobias Brockhoff

Aachen, April 22, 2025

Abstract

Process Mining is a field that addresses the challenge of analyzing event data generated by modern systems. By integrating techniques from data science and business process management, process mining enables the discovery, conformance checking, and enhancement of real-world processes based on event logs. However, process discovery algorithms often struggle with event data that contains noise, infrequent behaviour, outliers, or complex control flow. The presence of outliers in the event log often leads to the discovery of overly complex or imprecise models that are not the true representation of actual behaviour. This highlights the necessity and importance of preprocessing the event data before applying process discovery algorithms in order to ensure that the discovered models accurately reflect the underlying process. However, setting the hyperparameters for both preprocessing and process discovery is a non-trivial task and may significantly impact the quality of the resulting model.

This work introduces a *Meta-Discovery Algorithm* that utilizes a hyperparameter optimization framework based on *Multi-Objective Tree-Structured Parzen Estimator* [36] that uniquely considers the entire process discovery pipeline, both preprocessing and discovery steps, compared to existing approaches in this field. By tuning preprocessing and discovery parameters together, our approach ensures that event logs are refined before discovery, minimizing the impact of noise and infrequent behavior on the discovered models. We implement our approach as an extensible framework that integrates state-of-the-art methods for event data preprocessing, process discovery, and hyperparameter optimization. To achieve this we use the well-known toolkit ProM [41] for Process Mining and the powerful hyperparameter tuning framework Optuna [7].

We evaluate our method using real-life event logs, demonstrating that our optimization approach yields process models with consistently higher quality across the quality metrics: *fitness, precision, generalization*, and *simplicity*, compared to models discovered using default parameter settings.

Contents

List of Figures

List of Tables

1 Introduction

Process mining has emerged as a powerful approach for extracting insights from the vast volumes of event data generated by modern information systems [1]. As organizations increasingly digitize their operations, information systems continuously capture detailed records of business process executions, commonly stored as event logs. In the context of big data, process mining provides a bridge between raw event data and meaningful process knowledge. There exist three main types of process mining: *process discovery*, *conformance checking*, and *process enhancement* [1].

Process discovery aims to construct a process model from an event log automatically [5]. The goal is to uncover how a business process is actually executed based on observed behavior. The resulting process model provides a visual representation of the process. *Conformance checking* compares the behavior recorded in the event log with an existing process model [15]. *Process enhancement* uses the event log to improve or extend an existing model [1]. Among these, process discovery is the most fundamental and widely studied task [5]. It serves as the entry point for analyzing processes and lays the foundation for subsequent conformance and enhancement techniques.

A wide range of process discovery algorithms have been developed over the years, including the Alpha Miner [5], Heuristics Miner [44], Inductive Miner [27], ILP-based approaches [48], and Split Miner [9]. These algorithms differ in how they represent control-flow, handle noise, and balance model complexity. Since process models are constructed from event logs, the quality and structure of the log itself plays a crucial role in the effectiveness of the discovery algorithm. As a result, an algorithm that performs well on one log may yield poor results on another. Moreover, each algorithm has hyperparameters (e.g., filtering thresholds, dependency thresholds) that significantly influence its behavior and output. Yet these parameters are often set manually or remain at default values, leading to suboptimal or overly complex models. To handle issues related to noisy or inconsistent event data, preprocessing techniques are often applied before discovery. These include filtering infrequent behavior [38], repairing traces [39], or abstraction [21]. However, preprocessing introduces additional parameters that must be carefully picked to be effective.

This raises two critical problems: Which algorithm to pick for preprocessing or process discovery? How should the parameters of those algorithms be configured to achieve the desired results?

This thesis proposes a *Meta-Discovery Algorithm* (`MDA`) that addresses the challenges of algorithm selection, preprocessing, and hyperparameter tuning in process discovery. Given an event log, it automatically explores combinations of preprocessing methods, discovery algorithms, and parameter settings to generate process models reflecting trade-offs across all quality dimensions: *fitness, precision, generalization,* and *simplicity.*

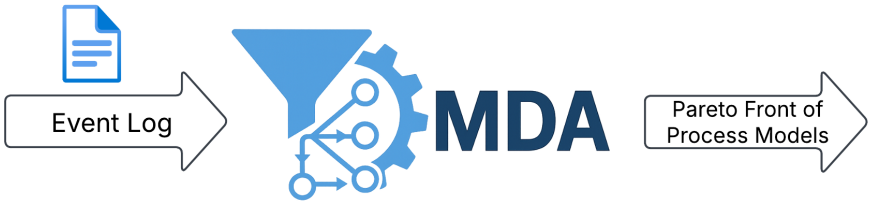


Figure 1.1: Meta-Discovery Algorithm for Process Mining

Our approach jointly optimizes preprocessing and process discovery using multi-objective hyperparameter tuning. It builds on the *Multi-Objective Tree-structured Parzen Estimator* (MOTPE)[36] from Optuna [7] and integrates ProM [41] components for process mining. The result is a *Pareto front* of diverse, non-dominated process models from which users can select the one that best meets their specific objectives.

We evaluate our approach on real-life event logs and demonstrate that it consistently identifies configurations that outperform default settings across all quality metrics.

1.1 Motivation

Although many process discovery algorithms are available, creating reliable and meaningful models from real-life event logs is still difficult. In practice, users often rely on trial-and-error, testing different combinations of preprocessing methods, discovery algorithms, and hyperparameter settings. This process is time-consuming and prone to mistakes. Some researchers have proposed recommendation systems that suggest suitable discovery algorithms based on the characteristics of the event log [24]. Others have explored the effect of tuning hyperparameters of discovery algorithms to improve model quality [8]. While these efforts address parts of the problem, they ignore the role of preprocessing before applying the discovery algorithm. Complex event logs can make process discovery algorithms slow or even infeasible. By reducing the size and complexity of the data through preprocessing, these algorithms can run more efficiently while also improving the quality of the process discovery results [37].

In addition, users may have different modeling goals. Some prioritize fitness to ensure all behavior is captured, while others may favor precision, simplicity, or generalization. These objectives often conflict, and a single fixed configuration is unlikely to satisfy them all.

To further illustrate the problem, we consider the results from prior research [24]. While we could not fully replicate the results due to implementation differences, our approach uses ProM, whereas the original study used PM4Py [10], we take their findings on the BPIC2012 [18] event log as a starting point. According to their recommendation, the best-performing algorithm for this event log is the Inductive Miner. We accept their reported performance metrics, but discover and visualize the Petri net using the Inductive Miner implementation in ProM, as shown in Figure 1.2.

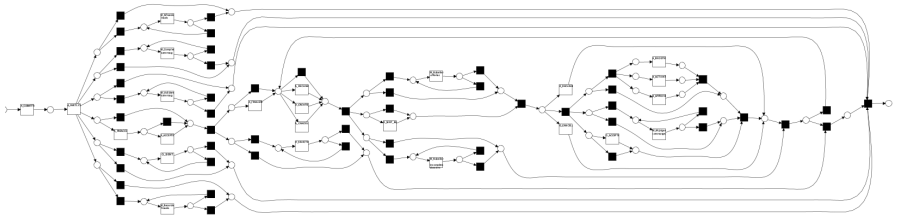


Figure 1.2: Recommended Algorithm: Fitness: 1.0, Precision: 0.11, Generalization: 0.96, Simplicity: 0.597070

Now, let us assume the user's primary goal is to discover a model with high *fitness*. The following model, shown in Figure 1.3, was discovered using our `MDA` approach. The only difference in this case is that our approach applied preprocessing before discovery. This example demonstrates that preprocessing can influence the outcome of process discovery. In this case, the resulting model is less complex while still maintaining high *fitness*.

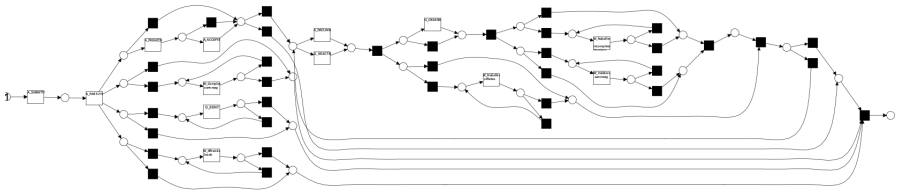


Figure 1.3: MDA: Inductive Miner (IM), Fitness: 0.94, Precision: 0.35, Generalization: 0.98, Simplicity: 0.90

Moreover, our approach can discover an entirely different configuration that better balances the quality metrics. As shown in Figure 1.4, the Split Miner algorithm, combined with preprocessing and tuned parameters, produces a model that scores highly across all dimensions.

These results confirm that preprocessing is essential in process discovery, directly impacting model quality.

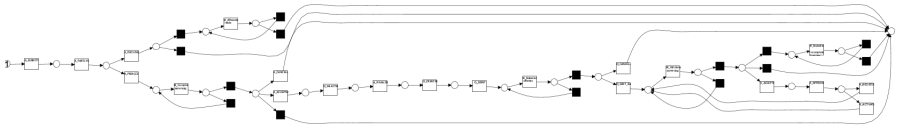


Figure 1.4: MDA: Split Miner, Fitness: 0.855, Precision: 0.955, Generalization: 0.98, Simplicity: 0.98

1.2 Problem Statement

While prior work has addressed algorithm recommendation and hyperparameter tuning independently, the joint impact of selecting and tuning both preprocessing and process discovery components remains largely overlooked. Currently, no general automated framework exists that optimizes the entire discovery pipeline across different quality dimensions.

1.3 Research Questions

This thesis investigates the following research questions:

RQ1: How can we formalize and implement an optimization approach that jointly tunes preprocessing and process discovery algorithms?

RQ2: How can we design the search space for our approach?

RQ3: How do different optimization strategies perform in our Process Mining setting?

RQ4: How does multi-objective hyperparameter tuning affect process model quality in terms of *fitness, precision, simplicity*, and *generalization*?

1.4 Research Goals

To answer these research questions and assess the effectiveness of `MDA`, we pursue the following goals:

- Develop a configurable, extensible pipeline for process discovery that supports hyperparameter tuning across preprocessing and process discovery stages.
- Benchmark multiple optimization strategies and search space structures.
- Evaluate `MDA` on diverse event logs to assess robustness and generalization.

1.5 Contributions

This thesis makes the following contributions:

- A formalization of process discovery as a multi-objective optimization problem over a configuration space structure that supports preprocessing and process discovery algorithm selection.
- An implementation integrating ProM and Optuna, with Docker-based deployment.
- A comprehensive empirical evaluation on real logs across all quality dimensions.
- A practical UI that facilitates configuration, monitoring, and analysis of optimization studies.

1.6 Thesis Structure

The remainder of this thesis is organized as follows: In Chapter 2 first present the general necessary notations and definitions. In Chapter 3 we position our approach next to recommendation algorithms and hyperparameter tuning of process discovery algorithms. In Chapter 4, we present our main contribution: the *Meta-Discovery Algorithm*. In Chapter 5, we provide the implementation approach of the proposed method. In Chapter 6, we describe our experimental setup and evaluation results.

2 Preliminaries

This chapter introduces mathematical notations and formal definitions used throughout this thesis. More detailed and additional formulations to some definitions that also consider hyperparameters will be introduced later in Chapter 4.

2.1 Event Data

Definition 2.1.1. (Events [1]) Let $\mathcal{E}$ be the set of all possible events. An event $e \in \mathcal{E}$ is a tuple $e = (c, a) \in \mathcal{U}_{case} \times \mathcal{U}_{act}$, where

- $\mathcal{U}_{case}$: The universe of all case identifiers (c), representing process instances.
- $\mathcal{U}_{act}$: The universe of all activity labels (a), representing tasks or actions performed. For technical reasons, in the following, we assume that $\mathcal{U}_{act}$ is countably infinite.

Let $e \in \mathcal{E}$ and n $\in \{case, act\}$, then $\#_n(e)$ denotes the value of the attribute n assigned to the event e.

Definition 2.1.2. (Cases [1]) A case corresponds to a unique process instance. Formally, for a given case identifier $c \in \mathcal{U}_{\text{case}}$, there is an associated trace:

$$\hat{c} = \langle e_1, e_2, \ldots, e_n \rangle$$

which is a non-empty, finite sequence of events, where for each $x \in \{1, \ldots, n\} : e_x \in \mathcal{E}$ and $\#_{case}(e_x) = c$.

Definition 2.1.3. (Finite Event Log [1]) A *simple trace* σ is defined as a finite sequence of activities over $\mathcal{U}^*_{act}$, i.e $\sigma \in \mathcal{U}^*_{act}$. Let $\mathcal{B}(\mathcal{U}^*_{act})$ denote the set of all finite multisets over $\mathcal{U}^*_{act}$. Then a *Finite Event Log* L is a finite multiset of traces, i.e $L \in \mathcal{B}(\mathcal{U}^*_{act})$.

A *Finite Event Log* can be represented as:

$$L = [\sigma_1^{(k_1)}, \sigma_2^{(k_2)}, \ldots, \sigma_n^{(k_n)}]$$

where each $\sigma_i \in \Sigma^*$ and $k_i \in \mathbb{N}^+$ denotes the number of times the trace σ_i occurs in the log.

3 Related Work

This chapter discusses the related work in the field that combines hyperparameter tuning with process discovery. The lack of research in this field has been noted by several authors [24],[14].

As already discussed, picking the right parameters for algorithms related to preprocessing and process discovery is a non-trivial task and may require significant effort. Poor parameter choices are the main issue when it comes to discovered process models that are too complex or too simplistic, thereby affecting fitness, precision and generalization. Thus, hyperparameter optimization is essential for process mining, as different configurations of preprocessing and discovery algorithms can produce significantly different results [8]. In the following, we explore and present the related work on this topic and compare the differences and similarities between our approach and existing approaches.

In [14], the author explores the effect of hyperparameter selection on the quality of the discovered process models and applies optimization techniques to improve the quality metrics: fitness, precision, and generalization. While it provides valuable insight into tuning discovery algorithms, it is limited only to process discovery, whereas our approach extends hyperparameter optimization to both preprocessing and discovery.

Another area of research closely related to our work is the automatic recommendation of process discovery algorithms. The approach in [24], utilizes meta-learning [42] to extract event log characteristics and predict the most suitable process discovery algorithm, achieving an accuracy of 92% in algorithm recommendations. This work contributes to the automated process discovery, but because of the static algorithm selection approach, it does not directly make use of hyperparameter tuning, so their recommendation is based on default settings of algorithms. In contrast, our approach defines a configurable process mining pipeline consisting of preprocessing, discovery, and conformance checking steps. We perform multi-objective hyperparameter optimization over this pipeline to improve the quality dimensions of the resulting process models.

Additionally, benchmarking studies [8] have shown the importance of hyperparameter tuning in process mining, but they do not directly propose a structured method/framework for hyperparameter optimization. Hyperparameter tuning is well-studied in Machine Learning, with techniques like Bayesian optimization, genetic algorithms, and reinforcement learning proving highly effective [20]. However, such techniques remain underutilized in the field of process mining, where most studies rely on manual tuning (trial-and-error) or heuristics-based configurations informed by default settings or domain knowledge. By combining ProM's [41] robust preprocessing and process discovery implementations with the state-of-the-art hyperparameter tuning framework Optuna [7], we enable automated tuning of process mining pipelines. This provides an extensible framework that optimizes process models across all quality dimensions.

4 Method

In this chapter, we provide an overview of the method developed in this thesis.

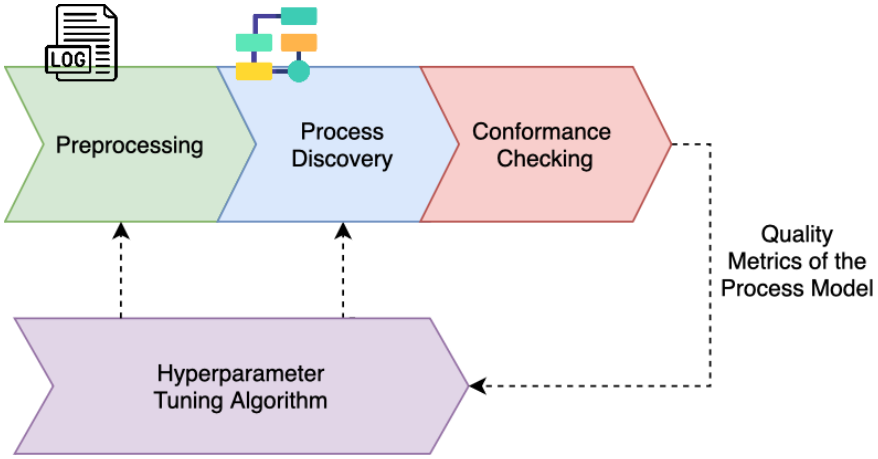


Figure 4.1: Overview of the `MDA` approach

As shown in Figure 4.1, our approach applies hyperparameter optimization to process discovery by tuning both the composition and parameters of the pipeline. The optimization works on two levels: (1) selecting which preprocessing steps and discovery algorithm to combine, and (2) tuning their associated hyperparameters. Each configuration is executed through the pipeline to discover a process model, which is then evaluated via conformance checking. The resulting quality metrics: *fitness*, *precision*, *simplicity*, *generalization*, and *F1-score* are used to guide the tuning algorithm toward better configurations in subsequent iterations.

We build on the definitions from Chapter 2 to describe each component of the pipeline and outline the role of multi-objective hyperparameter tuning in optimizing process discovery pipelines.

4.1 Preprocessing

The preprocessing step takes an event log as input and outputs a transformed event log. In this work, we focus on discovering a process model based on the activities in the input event log. Therefore, we do not consider preprocessing methods such as event abstraction [21] that modify the activity labels. We want to discover a process model for the *specific input log* and therefore require that the activity alphabet of the *preprocessed* event log remains unchanged. Most event log preprocessing functions require

some parameters that control to what extend the event log is modified. These parameters directly impact the structure of the preprocessed log and the quality of the resulting process model. In our approach, they are treated as hyperparameters and we include them in the optimization procedure.

Definition 4.1.1 (Parameterized Event Log Preprocessing Function)**.** A *Parameterized Event Log Preprocessing Function* is a function

$$\rho \colon X_\rho \to (\mathcal{B}(\mathcal{U}_{act}^*) \to \mathcal{B}(\mathcal{U}_{act}^*)) \,,$$

where $X_\rho = dom(\rho)$ is the hyperparameter space. For any $x \in X_\rho$, the function $\rho(x)$ transforms an event log into a preprocessed event log over the same alphabet.

While single preprocessing functions are sufficient for many scenarios, complex event logs may benefit from the *sequential application* of multiple preprocessing techniques [34]. To support this, we generalize the preprocessing definition to allow a sequence of k functions, where the output of each function serves as the input to the next. For the following definition, we use the notations from above.

Definition 4.1.2 (k-step Preprocessing Pipeline)**.** Let P be a set of parameterized event log preprocessing functions. A *k-step preprocessing pipeline* is a k-tuple $(\rho^1, \dots, \rho^k) \in P^k$. This pipeline induces a composed preprocessing function

$$\mathrm{Pre}_{(\rho^1,\dots,\rho^k)} \colon \left(dom(\rho^1) \times \cdots \times dom(\rho^k)\right) \to (\mathcal{B}(\mathcal{U}_{act}^*) \to \mathcal{B}(\mathcal{U}_{act}^*))$$

defined as

$$\mathrm{Pre}_{(\rho^1,\dots,\rho^k)}((x_1, \dots, x_k))(L) = \rho_{x_k}^k \left(\rho_{x_{k-1}}^{k-1} \left(\dots \rho_{x_1}^1(L) \dots\right)\right)$$

for any event log $L \in \mathcal{B}(\mathcal{U}_{act}^*)$, where $x_i \in dom(\rho^i)$.

4.2 Process Discovery

The next step of the pipeline of our algorithm is *Process Discovery*. As an input, it takes the *prepocessed* event log from the previous step. *Process Discovery* is a component of process mining that focuses on extracting process models from event logs.

Definition 4.2.1. (Process Model [3]) Let $\mathcal{U}_{PM}$ be the universe of process models. Given a model $M \in \mathcal{U}_{PM}$, $\mathcal{L}(M) \subseteq \mathcal{U}_{act}^*$ denotes its language, that is, the valid set of activity sequences (traces) according to the model.

Definition 4.2.2 (Parameterized Process Discovery Function)**.** A *Parameterized Process Discovery Function* is a function

$$d \colon X_d \to (\mathcal{B}(\mathcal{U}_{act}^*) \to \mathcal{U}_{PM}) \,,$$

where $X_d = \mathrm{dom}(d)$ is the hyperparameter space. For any $x \in X_d$, the function $d(x)$ maps an event log to a process model.

The discovery method operates on the preprocessed event log and constructs a process model that aims to reflect the behavior captured in the data. The quality of the resulting model depends on both the input log and the hyperparameters of the discovery algorithm. These hyperparameters influence key trade-offs between the accuracy in capturing the behavior recorded in the event and the complexity of the process model.

We incorporate a selection of distinct process discovery algorithms to capture a variety of modeling strategies and trade-offs. This enables the optimization process to adaptively select the most suitable algorithm for each event log. Rather than covering the entire landscape of discovery techniques, we focus on algorithms that are widely used in practice. A more detailed overview of the specific algorithms considered in our method is provided in Chapter 6.

Since our goal is to develop a general optimization framework that provides an analyst with a process model that balances all quality metrics, regardless of the characteristics of the input event log, we consider the discovery algorithm and its configuration as part of the hyperparameter space. This enables the hyperparameter optimizer to select a process discovery method and tune its respective hyperparameters for each iteration.

4.3 Process Model Quality

Conformance checking is a key concept in process mining that compares observed behavior in event logs with behavior allowed by a process model [15]. It identifies deviations and assesses how well a discovered model reflects the actual executions recorded in the data.

To evaluate the quality of a discovered process model, four main dimensions are typically considered in the process mining literature: *fitness*, *precision*, *simplicity*, and *generalization* [13]. In this work, we also consider the *F1-score*, defined as the harmonic mean of fitness and precision. It provides a single, balanced metric that reflects the trade-off between accurately replaying the event log and avoiding overly general behavior. Each of the quality dimensions captures a different aspect of model quality, and only when considered together can they provide a holistic view of how well a discovered model reflects the underlying process.

Fitness: Determines how well the process model can reproduce the behavior observed in the event log. Low fitness suggests that relevant behavior is missing from the model.

Precision: Assesses whether the model allows more behavior than what was observed in the event log. A precise model does not allow unnecessary or unrelated behavior that was never recorded in the event data.

Simplicity: Prefers models that are easy to understand and not overly complex, fewer arcs and transitions are considered better.

Generalization. A generalized model is not perfectly fitted to the specific instances in

the event log and can adapt to new but similar cases. A well-generalized model avoids overfitting to noise or rare occurrences in the data.

Balancing these dimensions is a core challenge in process discovery, as improvements in one metric often come at the expense of another [1]. For example, increasing fitness may reduce precision, while improving generalization might lead to more complex models and lower simplicity. Our optimization framework is designed to navigate these trade-offs and identify configurations that produce high-quality models tailored to each event log.

Several techniques have been proposed to quantify process model quality dimensions, particularly fitness and precision. Common approaches include token-based replay and alignments [4, 12]. Each of these has trade-offs: token-based replay is computationally efficient but tends to overestimate fitness, while alignments provide more accurate conformance checking at a higher computational cost. In this work, we adopt *alignment-based conformance checking* to evaluate both fitness and precision. Alignments offer precise, trace-level comparisons between log and model, distinguishing between behavior seen in the log but not allowed by the model (log moves) and behavior allowed by the model but not seen in the event log (model moves) [15]. Alignments are accurate and widely regarded as the standard for measuring conformance. Although alignment-based techniques are more computationally expensive, their precision and robustness make them especially suitable for our optimization framework.

Of course, a good process model should balance all four quality dimensions and none should be overlooked. However, many process discovery algorithms only focus on two or three of these aspects [13]. One reason for this imbalance is that fitness, precision, and generalization are more directly measurable through techniques such as alignments [15], making them easier to quantify and optimize. In contrast, simplicity is more abstract and often requires heuristic or subjective evaluation criteria, which complicates its integration into automated discovery pipelines.

4.4 Hyperparameter optimization

Hyperparameter tuning refers to the process of optimizing hyperparameters to improve the performance of a function/learning algorithm for a machine learning model. Hyperparameters are the parameters set before the execution of a function, unlike parameters that are derived during the execution. The goal in our scenario is to identify configurations that discover process models with high values across all quality metrics, including *fitness, precision, simplicity*, and *generalization*. In this work, we treat the *F1-Score* as part of the quality metrics to ease the comparison between the discovered process models.

Definition 4.4.1. (Multi-objective Hyperparameter Tuning [25]) Let Λ be the hyperparameter space, where each hyperparameter configuration is a vector: $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_n) \in \Lambda$ and λ_i denotes individual hyperparameter. Let $f : \Lambda \to \mathbb{R}^m$ be a multi-objective function that maps hyperparameter configurations to multiple performance metrics:

$$f(\lambda) = (f_1(\lambda), f_2(\lambda), \ldots, f_m(\lambda)).$$

The goal is to find the set of *Pareto-optimal* configurations $\Lambda^* \subseteq \Lambda$ such that no configuration in Λ dominates any $\lambda^* \in \Lambda^*$. Formally, a configuration $\lambda^* \in \Lambda$ is *Pareto-optimal* if there does not exist another configuration $\lambda \in \Lambda$ such that:

$$\forall j \in \{1, \ldots, m\}\ f_j(\lambda) \geq f_j(\lambda^*) \quad \text{and} \quad \exists j\ f_j(\lambda) > f_j(\lambda^*).$$

In our setting, each objective $f_1, \ldots, f_m$ corresponds to a quality metric evaluated on the discovered process model resulting from the hyperparameter configuration λ. Our goal is not only to optimize the hyperparameters of a given discovery algorithm, but also to select the most suitable algorithm itself. This introduces an additional decision layer: the algorithm choice determines which hyperparameters are relevant. As such, our problem can be translated into the *Combined Algorithm Selection and Hyperparameter problem*, where both the algorithm and its corresponding hyperparameters must be optimized.

Definition 4.4.2. (Combined Algorithm Selection and Hyperparameter Optimization (CASH) [40]) Formally, the *CASH* problem is defined as:

$$(A^*, \lambda^*) = \underset{A \in \mathcal{A},\ \lambda \in \Lambda_A}{\arg\max}\ f_A(\lambda),$$

where $\mathcal{A}$ is the set of candidate algorithms and Λ_A is the hyperparameter space associated with algorithm $A \in \mathcal{A}$.

A key point of the *CASH* formulation is that each configuration explores only a subset of the whole search space. The configuration consists of algorithm selection $A \in \mathcal{A}$ that determines the applicable hyperparameters $\Lambda_A \subset \Lambda$, and so it essentially restricts the search to a relevant conditional subspace. As a result, the configuration is constructed incrementally by starting with a categorical decision (e.g., algorithm), followed by sampling only the relevant hyperparameters. The hierarchical structure in our problem motivates the need for conditional search spaces. We can then formulate our *Hyperparameter Tuning for Process Discovery* problem as:

$$(\rho^*, d^*, x_\rho^*, x_d^*) = \underset{\rho \in P, x_\rho \in X_\rho, d \in D, x_d \in X_d}{\arg\max} f_{p,d}(x_\rho, x_d).$$

where $f_{p,d}$ is a multi-objective function, as defined in 4.4.1.

4.4.1 Flat vs. Conditional Search Spaces

In *traditional* search spaces [20], all hyperparameters are treated as equally relevant and active, regardless of context. Let $\Lambda_1, \ldots, \Lambda_n$ denote the domains of n individual hyperparameters. The flat configuration space is then defined as the *Cartesian product*:

$$\Lambda^{flat} = \Lambda_1 \times \Lambda_2 \times \cdots \times \Lambda_n.$$

This results in a fixed-dimensional, unconditional search space, where all hyperparameters are treated as active regardless of context. As a consequence, it may produce semantically invalid configurations. For instance, a configuration may specify a value for `noise_threshold` despite selecting an algorithm that does not support noise handling. In contrast, a *conditional search space* encodes dependencies between parameters and activates only those relevant in the current context.

Conditional Hyperparameter Space

To model conditional dependencies among hyperparameters, we adopt a tree-structured representation [31], where certain parameters are only active based on prior decisions. The search space Λ is then structured as a tree $\mathcal{T} = (V, E)$, where each node $v \in V$ represents a set of hyperparameters along with their associated domains. A directed edge $e = (v', v) \in E$ indicates a conditional dependency, meaning that the hyperparameters in node v are only active if a specific condition in the parent node v' is satisfied.

The overall search space Λ can be decomposed into K conditionally independent subspaces, where K is the total number of distinct root-to-leaf paths in the hierarchical tree $\mathcal{T}$:

$$\Lambda = \bigcup_{k=1}^{K} \Lambda_k,$$

Each subspace Λ_k corresponds to a flat search space of dimensionality $d_k \in \mathbb{N}$, determined by a specific path from the root to a leaf in $\mathcal{T}$. Each configuration in Λ_k activates the same set of hyperparameters, reflecting the same structural choices in the tree. This hierarchical structure enables the compact and expressive representation of search spaces with conditional dependencies. Based on the general formulation of conditional search spaces, in the next section, we define the structure of the search space used in our method, including both preprocessing and discovery components.

4.4.2 Hyperparameter Space for Meta Discovery Algorithm

To enable the joint optimization of preprocessing and discovery steps across multiple algorithm choices, we construct a conditional hyperparameter space that encodes both algorithm selection and their respective hyperparameters. Such a formulation aligns with the presented *Combined Algorithm Selection and Hyperparameter Optimization problem.*

To cleary represent this structure, we split the search space into two independent conditional trees: one for preprocessing and one for discovery. These are then combined to form complete pipeline configurations of our approach.

Preprocessing Search Space

Let P denote the set of available parametrized preprocessing functions. In Section 4.1 we defined a *k-step preprocessing pipeline.* In the current approach, we restrict this pipeline to contain at most two preprocessing steps, i.e., $k \in \{1, 2\}$. To allow for sequential

application of two different functions, we extend this set as:

$$P_{\text{seq}} = \{(\rho_1, \rho_2) \in P \times P \mid \rho_1 \neq \rho_2\}$$

The conditional hyperparameter search space for preprocessing, denoted Λ_{prep}, includes both single-function configurations and sequential combinations. While in the general hyperparameter tuning formulation, the algorithm is treated as a categorical decision, we instead represent the preprocessing function itself as part of the configuration to improve readability and better illustrate the structure of the search space, with the same idea of it being a categorical decision.

$$\Lambda_{\text{prep}} = \bigcup_{\rho \in P} \{\rho\} \times dom(\rho) \;\cup \bigcup_{(\rho_1, \rho_2) \in P_{seq}} \{(\rho_1, \rho_2)\} \times (dom(\rho_1) \times dom(\rho_2))$$

Figure 4.2 is an example of a *conditional* preprocessing search space. Depending on the choice of preprocessing configuration, either single or a sequential, different hyperparameters become active.

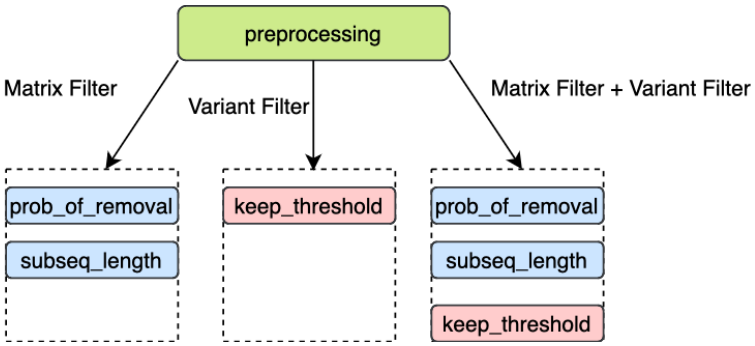


Figure 4.2: A tree-structured configuration space for preprocessing. Categorical decisions such as algorithm(s) activate different sets of valid hyperparameters.

Each branch defines a subspace Λ_k of structurally distinct configurations. In this example, $K = 3$, we have three distinct subspaces, each of them encompassing a different dimension size.

- $\Lambda_{\text{MatrixFilter}}$ with dimensionality $d_{\text{MatrixFilter}} = 2$, with active hyperparameters `prob_of_removal`, `subseq_length`.
- $\Lambda_{\text{VariantFilter}}$ with dimensionality $d_{\text{VaritantFilter}} = 1$, with active hyperparameter `keep_threshold`.
- $\Lambda_{\text{Matrix+Variant}}$ with dimensionality $d_{\text{Matrix+VariantFilter}} = 3$, with active hyperparameters `prob_of_removal`, `subseq_length`, `keep_threshold`.

Process Discovery Search Space

Similar to preprocessing, the discovery search space captures both the selection of a

discovery algorithm and the configuration of its associated hyperparameters. In this case, we also account for algorithms that support multiple variants, where each variant activates a distinct set of relevant hyperparameters.

Let $\mathcal{D}$ be the set of parameterized process discovery algorithms, and let V_d denote the set of variants associated with an algorithm $d \in \mathcal{D}$. As in the *preprocessing* case, we include the discovery function in the configuration tuple to better illustrate the structure of the conditional search space, even though they are treated as categorical decisions in practice. We define the search space for *process discovery* as follows:

$$\Lambda_{\text{disc}} = \bigcup_{\substack{d \in \mathcal{D}, \\ V_d = \emptyset}} \{d\} \times \text{dom}(d) \;\cup\; \bigcup_{\substack{d \in \mathcal{D}, \\ V_d \neq \emptyset}} \left(\{d\} \times \left(\bigcup_{v \in V_d} \{v\} \times \text{dom}(v) \right) \right)$$

This construction reflects a tree-structured search space, where the first categorical decision selects an algorithm $d \in \mathcal{D}$, and the second selects a variant $v \in V_d$ when the selected algorithm includes variants ($V_d \neq \emptyset$). Each variant then has its own hyperparameter space $\text{dom}(v)$. In practice, when an algorithm has variants, each variant corresponds to a structurally distinct function. For example, this is the case for the family of Inductive Miner [26] algorithms. However, we group such families under a single categorical decision (e.g., "Inductive Miner") to avoid introducing sampling bias towards algorithms with many variants. Without this grouping, algorithms with many variants would be overrepresented in the search space and perhaps more likely to be selected by the optimizer.

The right-hand side of the formula captures this two-step structure: a categorical choice of algorithm, followed by a categorical choice of variant. This part of the formulation can be omitted without loss of generality by treating each variant as a separate discovery function in $\mathcal{D}$.

Figure 4.3 illustrates an example of a *conditional* process discovery search space. Depending on the choice of process discovery function (or a specific variant), different parameters become active.

- $\Lambda_{\text{SplitMiner}}$ with dimensionality $d_{\text{SplitMiner}} = 2$, with active hyperparameters `eta` and `epsilon`.
- The subspace $\Lambda_{\text{IM\_default}}$ has dimensionality $d_{\text{IM\_default}} = 0$, as it contains no tunable hyperparameters. Its only valid configuration is the categorical choice itself (i.e., the *conditional decisions* defining the path).
- $\Lambda_{\text{IM\_infrequent}}$ with dimensionality $d_{\text{IM\_infrequent}} = 1$, with active hyperparameter `noise_threshold`.

With both conditional search spaces for *preprocessing* and *process discovery* defined, we now combine them into a unified conditional search space, which serves as the foundation for our optimization approach. This combined search space, denoted by Λ_{MDA}, consists of tuples that represent complete pipeline configurations. Each tuple specifies

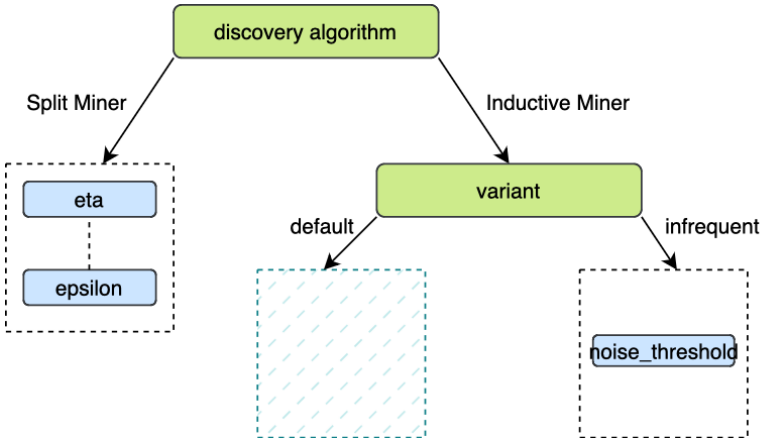


Figure 4.3: A tree-structured configuration space for process discovery. Categorical decisions such as algorithm and variant activate different sets of valid hyperparameters.

either one or two preprocessing steps, a discovery algorithm, and the corresponding hyperparameters for each component. For simplicity, we do not include discovery algorithms with variants.

$$\begin{aligned}\Lambda_{\text{MDA}} = \; & \{(\rho,\ X_\rho,\ d,\ X_d) \mid (\rho,\ X_\rho) \in \Lambda_{\text{prep}},\ (d,\ X_d) \in \Lambda_{\text{disc}}\} \\ \cup \; & \{(\rho_1,\ \rho_2,\ X_{\rho_1},\ X_{\rho_2},\ d,\ X_d) \mid \\ & \qquad ((\rho_1,\ \rho_2),\ X_{\rho_1},\ X_{\rho_2}) \in \Lambda_{\text{prep}},\ (d,\ X_d) \in \Lambda_{\text{disc}}\}\end{aligned}$$

A configuration using a single preprocessing function is applied to an event log as follows:

$$(\rho,\ x_\rho,\ d,\ x_d) \overset{L}{\rightsquigarrow} d_{x_d}(\rho_{x_\rho}(L)), \quad \text{where } L \in \mathcal{B}(\mathcal{U}^*_{\text{act}}) \text{ is an event log.}$$

In the case of two sequential preprocessing steps, the application is defined as:

$$(\rho^1,\ \rho^2,\ x_{\rho^1},\ x_{\rho^2},\ d,\ x_d) \overset{L}{\rightsquigarrow} d_{x_d}(\text{Pre}_{(\rho^1,\rho^2)}(x_{\rho^1}, x_{\rho^2})(L)))$$

Each configuration lies in a subspace $\Lambda \subseteq \Lambda_{\text{MDA}}$. The dimensionality is determined by structural choices (e.g., selected functions, presence of variants). By activating only relevant parameters, the conditional setup avoids semantically invalid configurations. Such a conditional formulation improves the efficiency of the search process by reducing the dimensionality of each subspace and guiding the optimizer to meaningful regions of the search space [30, 31].

4.5 Meta-Discovery Algorithm for Process Mining

This section formalizes our proposed *meta-level* optimization procedure, which integrates all previously described components: preprocessing, process discovery, conformance checking, and multi-objective hyperparameter tuning. The optimization follows a black-box strategy that iteratively explores a conditional search space, evaluates generated process models, and records quality metrics. The search space is explored using Optuna, which provides a range of state-of-the-art multi-objective optimization algorithms, including NSGA-II [16], NSGA-III [17, 23], and Multi-Objective TPE [36]. These algorithms, referred to as *samplers* in Optuna, are responsible for selecting promising configurations during the optimization process. The optimization process in `MDA` continues for a fixed number of trials N. A trial marks a configuration sampled from the search space Λ_{MDA} by the sampler. This includes one or two preprocessing functions, a discovery algorithm, and all relevant hyperparameters. The discovered model is evaluated using alignment-based conformance checking. The resulting metrics serve as objective values to guide the optimizer in sampling better configurations in subsequent iterations. Trials that result in timeouts or undefined metrics are pruned. Pruning is another important feature of Optuna, it refers to the ability to terminate unpromising trials early based on intermediate objective values. This helps accelerate the optimization process by avoiding unnecessary computation and using partial results to inform future sampling. However, in the *multi-objective* setting, pruning is not fully supported. Terminated trials are simply discarded without providing any feedback to the sampler. In effect, it is as if the trial never occurred. We examine the practical consequences of this limitation in Chapter 6. All evaluated configurations are stored, and at the end, a set of *Pareto-optimal* process models $\mathcal{M}_{\text{best}} \subseteq \mathcal{M}$ is returned.

For readability, the following Algorithm 1 shows only one configuration form. In practice, the sampler handles multiple valid configuration structures, depending on whether the pipeline includes one or two preprocessing steps and whether the discovery algorithm has variants.

Algorithm 1 Meta-Discovery Optimization Approach

Hyperparameter sampler $\mathcal{O}$, Conditional configuration space Λ_{MDA}

Input: Event log $L \in \mathcal{B}(\mathcal{U}_{act}^*)$
Budget: number of trials N, timeouts T_{disc}, T_{align}

Output: Set of evaluated models $\mathcal{M}$; Pareto-optimal subset $\mathcal{M}_{best}$

Initialize empty result set $\mathcal{M} \leftarrow \emptyset$
for $i = 1$ to N **do**
  Sample configuration $\lambda_i = (\rho_i, x_{p_i}, d_i, x_{d_i}) \leftarrow \mathcal{O}.\texttt{suggest\_parameter}(\Lambda_{\text{MDA}})$
  Apply preprocessing: $L_i \leftarrow \rho_i(x_{p_i})(L)$
  Attempt discovery $M_i \leftarrow d_i(x_{d_i})(L_i)$ within timeout T_{disc}
  if discovery fails or times out **then**
    $\mathcal{O}.\texttt{prune}(\lambda_i)$ ▷ Trial failed—invalid process model
    continue
  end if
  Attempt evaluation of M_i on L within timeout T_{align}
  if evaluation fails or any $f_j(\lambda_i)$ is undefined **then**
    $\mathcal{O}.\texttt{prune}(\lambda_i)$ ▷ Trial failed—invalid quality metrics
    continue
  end if
  $\mathcal{O}.\texttt{update\_with\_observation}(\lambda_i, f(\lambda_i))$
  $\mathcal{M} \leftarrow \mathcal{M} \cup \{(M_i, f(\lambda_i))\}$
end for
Extract Pareto-optimal subset $\mathcal{M}_{best} \subseteq \mathcal{M}$
return $\mathcal{M}_{\text{best}}$

5 Implementation

In this chapter, we show the design decisions of the *Meta-Discovery Algorithm*. First, in Section 5.1 we present the overall design and then dive into the implementation details of each major component.

The project containing the algorithm presented in this thesis can be accessed at `https://github.com/georgigrgv/MetaDiscoveryThesis`, where usage instructions can be found as well.

5.1 Architecture

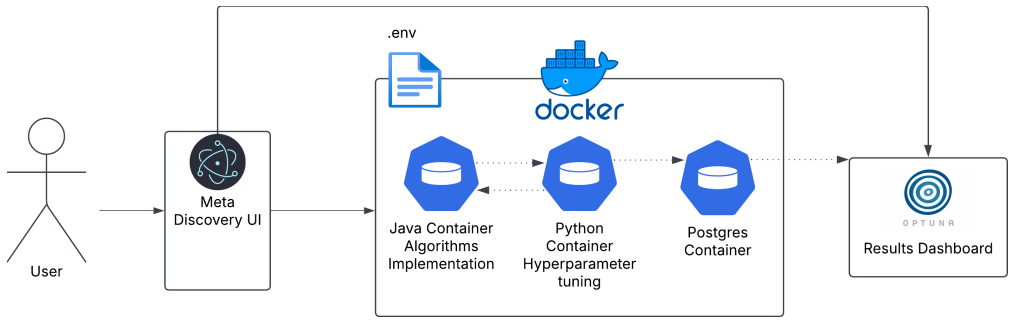


Figure 5.1: Architecture Overview

The *Meta-Discovery Algorithm* (`MDA`) is designed as a framework to automate and optimize the entire process mining pipeline. It is structured into three layers: *User Interface (UI)* in Section 5.3, *Meta Discovery Algorithm* in Section 5.2, and a *Dashboard* for the results of our algorithm. The UI serves as an entry point, allowing users to run experiments without manually selecting algorithms, tuning hyperparameters, or understanding the underlying optimization logic, as it defaults to the best-performing setup found through our evaluation. One of the key architectural challenges in this project was bridging the gap between the Java-based ProM [41] ecosystem and the Python-based optimization framework - Optuna [7]. This motivated the need for Docker [1]. The core layer encapsulated in Docker containers selects configurations, executes the process discovery pipeline, and stores the resulting quality metrics and models. Finally, the dashboard, provided by Optuna, shows the optimization results.

[1] `https://www.docker.com`

5.2 Meta-Discovery Algorithm

As illustrated in the Figure 5.2, our *Meta-Discovery Algorithm* consists of two processes, each encapsulated in a separate Docker container. `Java Process`, responsible for preprocessing event logs, applying discovery algorithms, and performing conformance checking, and a `Python Process` used for hyperparameter tuning. The event log is loaded and managed by the Java Process. At each trial, the Python Process suggests a configuration, which the Java Process executes using the specified hyperparameters. The resulting quality metrics are sent back to the Python Process to guide future sampling and are stored together with the associated configuration in a PostgreSQL [2] database.

In the following section, we present the implementation decisions and chosen frameworks inside the Meta Discovery Algorithm.

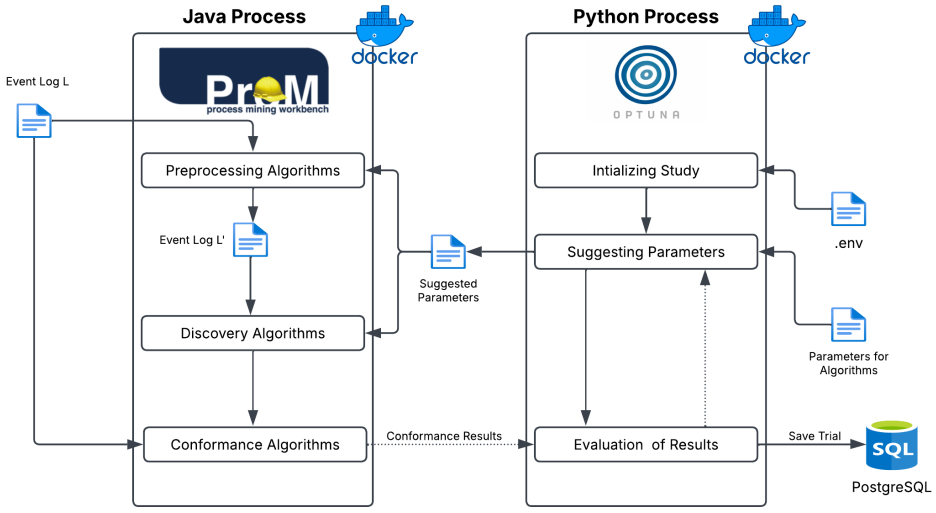


Figure 5.2: Meta-Discovery Algorithm Overview

5.2.1 Design Choices

This section outlines the important design choices and decisions for the implementation of the *Meta-Discovery Algorithm*. We treat the *Meta-Discovery Algorithm* as a black-box approach, therefore we do not go into detail on the implementation of specific parts of the algorithm, rather we will present an overview of the structure implemented and the design decisions made.

First, as illustrated in Figure 5.2 and mentioned in Section 5.2 our algorithm uses two processes. The design decisions were influenced by the choice of the ProM framework for Process Mining written in Java, and the powerful optimization libraries available in Python, e.g., Optuna.

[2] `https://www.postgresql.org`

`ProM`: ProM offers reliable implementation of a wide range of preprocessing, process discovery, and conformance checking techniques. Since our goal is to evaluate the importance of hyperparameter tuning on process discovery pipelines, this framework makes a perfect candidate by offering us algorithms and their variants whose implementation can not always be found elsewhere.

`Optuna`: The choice for Optuna was influenced by the large active community, and state-of-the-art algorithms providing support for conditional search spaces and multi-objective hyperparameter tuning. Lastly, their integrated dashboard makes it easy to get insights on finished trials by showing the influence of hyperparameters, their relationship, and correlation between the results in our multi-objective environment.

In order to achieve consistent runtime behavior and avoiding environment-related issues, we decided to opt for Docker. This approach eliminates dependency conflicts and provides reproducible execution across different systems. Docker's lightweight virtualization introduces minimal overhead, making it a practical choice for our framework. Additionally, Docker provides database management by allowing host directories to be mounted as volumes. This setup allows Optuna's dashboard to visualize both previous and ongoing studies via a running database.

The communication between the two containers is developed through a REST API. The Java Process exposes an HTTP service, that is based on Spark [3], which is accessed by the Python Process running the optimization study. This process sends POST requests containing trial configurations encoded in JSON [4] format. These configurations include different combinations of preprocessing methods, discovery algorithm variants, and their respective hyperparameters. The Java Process executes the pipeline and returns the evaluated quality metrics: *fitness, precision, F1-score, simplicity*, and *generalization* back to the Python Process as a JSON response.

Before the optimization begins, a configuration file containing all available algorithms, their possible variants, and the corresponding hyperparameter ranges must be provided. Optuna uses this file to construct the search space from which it samples trial configurations. Additionally, the structure of the file is designed to facilitate integration of new algorithms.

Table 5.1 shows the preprocessing, process discovery, and conformance checking algorithms that we have currently integrated into our framework. In addition to the plugins listed, we adopted the Split Miner [5] implementation. For simplicity and for generalization, we used the formulas presented in [13].

[3] `https://spark.apache.org/docs/latest/monitoring.html#rest-api`
[4] `https://www.json.org/json-en.html`
[5] `https://github.com/iharsuvorau/split-miner`

Table 5.1: Overview of ProM's Plugins Used in the Pipeline

Preprocessing
`Matrix Filter`: by Mohammadreza (FaniSani@pads.rwth-aachen.de).
`Repair Event Log!`: by Mohammadreza (FaniSani@pads.rwth-aachen.de).
`Variant Filter`: by Mohammadreza (FaniSani@pads.rwth-aachen.de).
`Filter Event Log`: by D.Fahland (d.fahland@tue.nl).
Process Discovery
`Alpha Miner`: by S.J. van Zelst, B.F. van Dongen, L.M.A. Tonnaer (s.j.v.zelst@tue.nl).
`Mine Petri net with Inductive Miner`: by S.J.J. Leemans (s.j.j.leemans@tue.nl).
`Mine for a Heuristics Net using Heuristics Miner`: by A.J.M.M. Weijters (a.j.m.m.weijters@tue.nl).
`ILP-Based Process Discovery`: by S.J. van Zelst (s.j.v.zelst@tue.nl.
Conformance Checking
`Replay a Log on Petri Net for Conformance Analysis`: by Arya Adriansyah (a.adriansyah@tue.nl).
`Check Precision using Escaping Edges`: by H.M.W. Verbeek (h.m.w.verbeek@tue.nl).
`Anti-Alignment Precision/Generalization`: by Boudewijn van Dongen (b.f.v.dongen@tue.nl)
Miscellaneous
`Convert Heuristics net into Petri net`: by J.T.S. Ribeiro (j.t.s.ribeiro@tue.nl).
`Create Accepting Petri Net`: by H.M.W. Verbeek (h.m.w.verbeek@tue.nl).
`Reduce Silent Transitions, Preserve Behavior`: by H.M.W Verbeek (h.m.w.verbeek@tue.nl).

5.3 User Interface (UI)

The User Interface is implemented as an *Electron*[6] application with a *React*[7] frontend and a *Node.js*[8] backend. Electron enables cross-platform deployment by combining web technologies with system-level access. The backend orchestrates the Docker containers.

The goal of the *Meta-Discovery UI* is to simplify the configuration, execution, and monitoring of process discovery experiments. It enables users to focus on analyzing results rather than managing complex setups. The only requirement for running the application is a Docker installation, as all data processing and computations are performed inside Docker containers, removing the need for manual dependency management. The UI offers an intuitive way to start and stop experiments and visualize results. Figure 5.3 shows the interface before a study is initiated, while Figure 5.4 displays the active study screen, including available monitoring and control options.

[6] `https://www.electronjs.org`
[7] `https://react.dev`
[8] `https://nodejs.org/en`

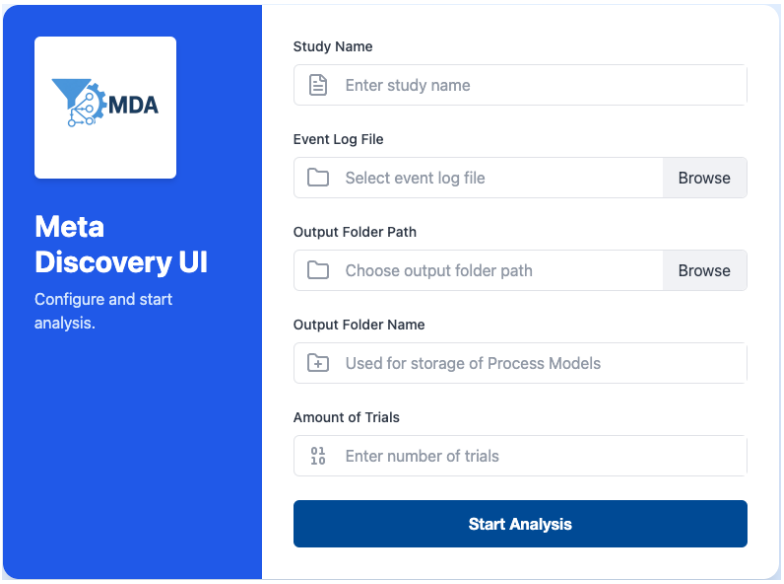


Figure 5.3: Configuration of a study

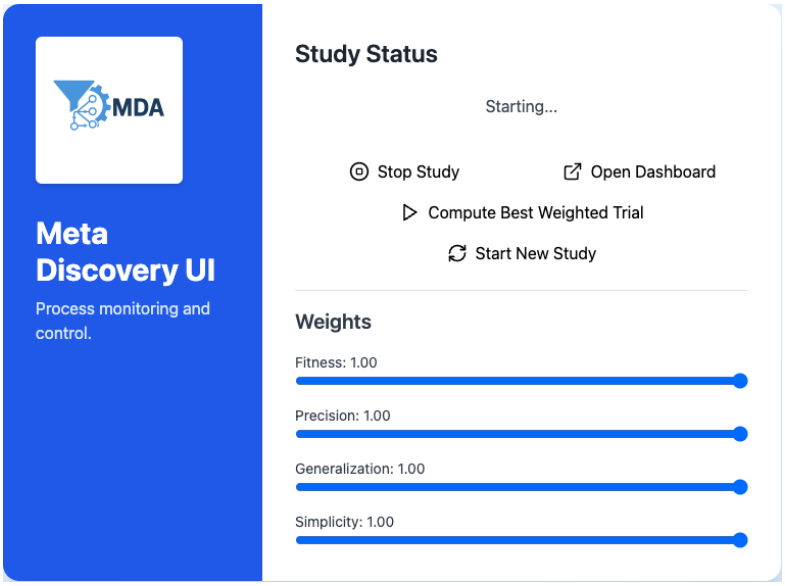


Figure 5.4: Running study

To initiate a study, the user provides the required input through the configuration form shown in Figure 5.3. Upon clicking the Start Analysis button, the UI sends a request to the *Electron* main process to generate a configuration file *(.env)* based on the user inputs. This file is then used to launch the Docker containers. Once the containers are successfully started, the UI navigates to the process screen 5.4 where progress mon-

itoring and control options are available.

From this screen, the user can:

- STOP STUDY: stops all Docker containers except for the database container. The user can still access the study results
- OPEN DASHBOARD: spawns an instance of Optuna's dashboard and opens it in the user's default browser.
- START NEW STUDY: stops all containers and returns the user to the main screen to begin a new study.
- COMPUTE BEST WEIGHTED TRIAL: calculates a score for each trial based on the defined quality metric weights and returns the trial number with the highest score.

Example of an interaction with the UI is summarized in the UML sequence diagram in Figure 5.5. For simplicity, two internal operations have been omitted from the diagram and are marked with red labels [9],[10].

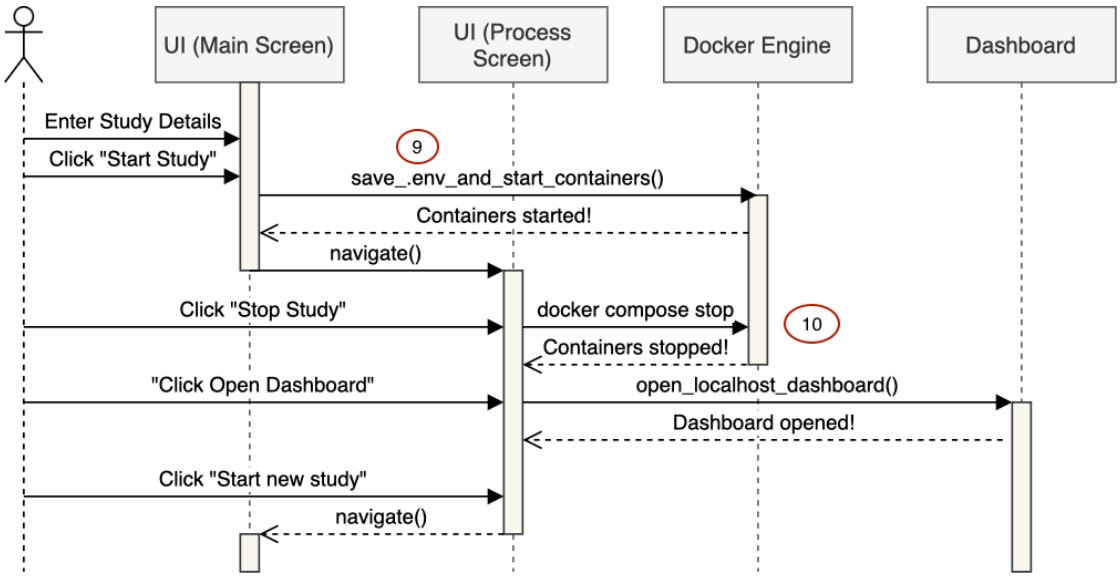


Figure 5.5: Example Workflow

[9] In practice, the system also validates the *(.env)* file contents and checks Docker availability before execution.

[10] STOP STUDY does not stop all containers as shown in the diagram.

6 Evaluation

We structure the evaluation of our `MDA` approach around three main objectives and their related questions:

- `Picking the optimal hyperparameter sampler`
  - Which one is most suitable for the method we proposed in Chapter 4
- `MDA vs. Baseline`
  - Did we make an improvement in process model quality over default configurations, and is it significant?
- `Assess the effect of pruning and incorporating preprocessing`
  - How does pruning affect the sampler?
  - How important is preprocessing?

This chapter is structured as follows. Section 6.1 introduces the experimental setup, including hardware and environment specifications, and descriptions of the selected event logs. Section 6.2 details the preprocessing and discovery methods included in the search space, along with their hyperparameters. Subsequent sections present the evaluation results, comparative analyses of configurations, and a discussion of key insights from the experiments.

6.1 Experimental Setup

We performed our benchmark on a 10-core Apple M1 Pro CPU @ 3.2GHz with 16GB RAM running Java 8 and Python 3.11 in the two containers we presented. For Docker we allocated 9 CPU cores and 14 GB of memory to the runtime environment. A maximum heap size of 8 GB was assigned to the Java Virtual Machine.

For our evaluation, we selected 5 distinct real-life event logs from diverse domains, including healthcare, finance, and IT service management. They are presented in the Table 6.1.

By including logs of varying size and complexity, we aim to assess how well our algorithm generalizes and performs under different levels of process variability and noise. The complexity of an event log can be characterized by several factors. A log is typically considered complex when it has high behavioral diversity (i.e., a high ratio of distinct traces), a large number of activities, and wide variation in trace lengths. For example, the *SEPSIS* log, despite being relatively small in size, has a very high percentage of

Table 6.1: Overview of the event logs used in the evaluation.

Log Name	Traces	Distinct Traces	Events	Event Classes	Trace Length		
					Min	Avg	Max
RTFMP [29]	150,370	0.2%	561,470	11	2	4	20
BPIC2017 [19]	31,509	50%	1,202,267	66	10	38	180
BPIC2012 [18]	13,087	33%	262,200	36	3	20	175
HOSPITAL [33]	100,000	1%	451,359	18	1	5	217
SEPSIS [32]	1,050	80.6%	15,214	16	3	14	185

distinct traces (80.6%), indicating high variability in behavior and a likely presence of unique cases, which highlights its structural complexity [1]. Similarly, *BPIC2017* contains a large number of traces and activities, combined with long and varying trace lengths, making it challenging to model accurately.

6.2 Algorithm Configurations

This section presents the algorithms selected for our evaluation and their respective configurations. We focus on both preprocessing techniques in Table 6.2 and process discovery algorithms in Table 6.3. We include diverse approaches to ensure comprehensive analysis. For the hyperparameter optimization, we employ *Optuna* with the samplers detailed in Table 6.4.

6.2.1 Preprocessing Algorithms

To improve the quality of discovered process models and reduce noise in event logs, we include four preprocessing techniques, each targeting a different aspect of log refinement. These filters are selected to balance the removal of outlier behavior with the preservation of meaningful process variants.

In Table 6.2, we present the ranges for the hyperparameters for all preprocessing algorithms we use. Important to note is that for classifier we always use *Event Name* to maintain a uniform representation of behavior across all event logs and compatibility with the discovery and conformance checking components of our pipeline.

- **Matrix Filter** [38]: Removes traces with unlikely behavioral patterns based on conditional probabilities over subsequences. For each trace, all subsequences of a given length (`subsequence_length`) are evaluated. If the probability of the next activity falls below a threshold (`prob_of_removal`), the trace is considered anomalous and removed.

- **Repair Event Log!** [39]: Replaces low-probability activity subsequences with more frequent alternatives based on their surrounding context. It uses a sliding window of configurable length (`subsequence_length`) and a probability threshold (`prob_of_removal`) to detect and repair outlier behavior without removing traces.

- **Variant Filter**: Removes infrequent trace variants by retaining the most n frequent variants such that their cumulative probability covers a specified percentage of the log (`threshold_frequency`).
- **Project Traces**: Removes infrequent events across all traces based on a frequency threshold (`threshold_frequency`), keeping only those that occur often enough.

Table 6.2: Preprocessing algorithms and their hyperparameters.

Algorithm	**Hyperparameter**	**Range**
Matrix Filter [38]	prob_of_removal subsequence_length	[0.0, 0.25] [1, 3]
Repair Event Log! [39]	prob_of_removal subsequence_length	[0.0, 0.25] [1, 3]
Variant Filter	threshold_frequency	[0.0, 1.0]
Project Traces (Event frequency)	threshold_frequency	[0.0, 1.0]

Our initial implementation used the full range of hyperparameter values for the *Matrix Filter* and *Repair Event Log* algorithms. However, we found that certain values outside specific subranges led to errors or invalid outputs. As a result, we limited the considered value ranges to those that produce valid results.

6.2.2 Process Discovery Algorithms

Table 6.3 lists the process discovery algorithms evaluated in our study. With this, we aim to cover a representative set of discovery techniques commonly used in practice and research.

We include Alpha Miner [5] as a foundational method based on direct-follows relations. Despite its simplicity, it is known to struggle with noisy or incomplete logs. The Flexible Heuristics Miner [45] enhances the classic Heuristics Miner with advanced heuristics and configurable thresholds, improving its ability to handle noise. It is designed for real-life event logs with complex and less structured behavior. The Inductive Miner [26] is a block-structured process discovery algorithm that recursively detects control-flow cuts in the directly-follows graph, constructed from the event log, to build sound and structured process models. One of the key strengths of the Inductive Miner lies in its formal guarantees regarding *fitness*. Split Miner [9] is a discovery algorithm that constructs accurate and simple process models by filtering the directly-follows graph and detecting concurrency through split gateway analysis. It guarantees deadlock-free models and a balance between *fitness* and *precision*. Finally, the [48] Hybrid ILP Miner was included for its use of hybrid regions, which extend classical ILP-based process discovery to more efficiently capture complex workflow patterns.

We intentionally selected algorithms with varying levels of structural bias (e.g., Petri nets, Process trees, BPMN), noise robustness, and hyperparameter complexity to evaluate the effectiveness of our tuning framework under diverse conditions. Wherever applicable, we evaluated multiple variants of the same algorithm to capture their performance across different parameterizations.

Table 6.3: Process discovery algorithms and their hyperparameters.

Algorithm	Variant	Hyperparameter	Range
Alpha Miner	Classic [5]	–	–
	Plus [35]	–	–
	Plus_Plus [46]	–	–
	Sharp [47]	–	–
	Robust	causality_threshold	[0, 100]
		noise_frequency (LF)	[0.0, 1.0]
		noise_threshold (MF)	[0.0, 120.0]
Inductive Miner	Default (IM) [26]	–	–
	Infrequent (IMf) [27]	noise_threshold	[0.0, 1.0]
	All operators (IMa) [26]	–	–
	Infrequent & All operators (IMfa) [26]	noise_threshold	[0.0, 1.0]
	Incompleteness (IMc) [26]	–	–
	Exhaustive K-successor (IMk)	–	–
	Life Cycle (IMlc) [26]	–	–
	Infrequent & Life Cycle (IMflc) [26]	noise_threshold	[0.0, 1.0]
Flexible Heuristics Miner [45]		dependency_threshold	[0.0, 1.0]
		relative_to_best	[0.0, 1.0]
		length_one_loops	[0.0, 1.0]
		length_two_loops	[0.0, 1.0]
		long_distance_threshold	[0.0, 1.0]
		long_distance_dependency	{true, false}
		consider_self_loops	{true, false}
		all_tasks_connected	{true, false}
Split Miner [9]		eta	[0.0, 1.0]
		epsilon	[0.0, 1.0]
		parallelism_first	{true, false}
		replace_iors	{true, false}
		remove_loop_activities	{true, false}
Hybrid ILP Miner [48]		lp_objective	{Minimize Arcs, Unweighted Parikh values, Weighted Parikh values (abs), Weighted Parikh values (rel)}
		optional_constraint_pnet	{true, false}
		lp_filter	{None, Sequence Encoding Filter, Slack Variable Filter}
		slack_variable_filter_threshold	[0.0, 1.0]
		sequence_encoding_cut_off_level	[0.0, 1.0]
		lp_variable_type	{Two variables per event, One per event (loop), One per event}
		discovery_strategy	{Random, Alpha, Heuristics, Fuzzy, Standard, Mini, Midi, Maxi, Average, Directly Follows}

6.2.3 Hyperparameter Samplers

To perform hyperparameter tuning, we use Optuna's built-in samplers, each implementing a distinct search strategy for exploring the parameter space. Table 6.4 summarizes their key characteristics.

Table 6.4: Overview of Optuna samplers.

Sampler	Multi-objective	Pruning Support	Search Space
Random	Yes	Yes	Flat / Conditional
MOTPE [36]	Yes	No	Flat / Conditional
NSGA-II [16]	Yes	No	Flat
NSGA-III [17, 23]	Yes	No	Flat

Rather than analyzing the internal mechanics of each algorithm in depth, we focus on evaluating their practical effectiveness in our specific setting. In particular, we aim to assess how well each sampler navigates the conditional search space that we defined in Chapter 4, and how effectively it handles our multi-objective optimization problem.

The Random sampler serves as our baseline, selecting configurations uniformly at random without using feedback from previous evaluations. In contrast, the Multi-Objective Tree-Structured Parzen Estimator (`MOTPE`) is a Bayesian Optimization approach. It extends the well-known Tree-Structured Parzen Estimator (`TPE`) to handle multiple objectives and complex search spaces. `MOTPE` [36] uses Pareto dominance to distinguish between promising and less promising observations, thereby guiding the search toward configurations that achieve better trade-offs across all objectives. On the other hand, `NSGA-II` [16] and `NSGA-III` [17, 23] are both evolutionary algorithms specifically designed for multi-objective optimization. These methods use non-dominated sorting to rank solutions and maintain a diverse population throughout the search. `NSGA-II` ensures diversity using crowding distance, whereas `NSGA-III` introduces reference points, which are particularly effective in complex multi-objective optimization problems.

Finally, although Optuna states that all samplers support both flat and conditional search spaces, some perform less efficiently in conditional settings [1].

6.3 Sampler Performance Analysis

To ensure efficiency and consistency, we applied a 5-minute timeout for both process discovery and conformance checking, preventing infinite execution. Trials resulting in errors were pruned and excluded from the studies. For discovery algorithms that do not natively produce Petri nets (e.g., Heuristic Miner, Split Miner), we used ProM's recommended conversion plugins. All conformance checking was performed using ProM's default plugin settings. For our multi-objective function, we use *fitness, precision, generalization* and *simplicity*. We also include *F1-Score* to later simplify comparison across process models. For the samplers in Table 6.4, we use the default settings. However, for `MOTPE`, we explicitly set the `multivariate` and `group` flags, as recommended by Optuna. Multivariate sampling helps the algorithm better capture dependencies among hyperparameters, while the group flag decomposes the search space into subspaces based on the conditional structure, which aligns with our formal method presented in Chapter 4. To

[1] `https://optuna.readthedocs.io/en/stable/reference/samplers/index.html`

evaluate and compare the performance of different samplers and space structures, we use the *hypervolume* metric [6]. It measures how much of the objective space is dominated by the *Pareto front*, capturing both their overall quality and how well they cover trade-offs across objectives like *fitness, precision, generalization*, and *simplicity*. Since the objective metrics in this context are within the range $[0, 1]$, and we try to maximize them, a reference point of 0 is used. A larger hypervolume indicates that the *Pareto front* includes models that perform strongly across a diverse range of these objectives.

In Optuna, a `study` corresponds to an optimization task, i.e., a set of trials. We started each study with 100 trials, and we increased this number to evaluate performance scalability across samplers and event logs.

Table 6.5: Samplers performance in a conditional search space (100 trials).

Event Log	Sampler	Hypervolume	Time to Finish	Failed Trials
SEPSIS	MOTPE	0.704	30m	30
	NSGA-III	0.664	40m	30
	NSGA-II	**0.708**	37m	26
	Random	0.667	30m	36
HOSPITAL	MOTPE	**0.968**	27m	26
	NSGA-III	0.967	16m	37
	NSGA-II	0.950	43m	53
	Random	0.934	25m	45
RTFMP	MOTPE	**0.985**	8m	24
	NSGA-III	0.975	13m	40
	NSGA-II	0.977	7m	37
	Random	0.966	6m	27
BPIC2012	MOTPE	**0.701**	1h 49m	42
	NSGA-III	0.678	1h 22m	54
	NSGA-II	0.678	45m	51
	Random	0.666	1h 22m	62
BPIC2017	MOTPE	0.524	2h 20m	54
	NSGA-III	**0.527**	2h 19m	55
	NSGA-II	0.501	2h 30m	56
	Random	0.374	2h 10m	56

Table 6.6: MOTPE performance in a conditional search space (300 trials.)

Event Log	Hypervolume	Time to Finish	Failed Trials
SEPSIS	0.781	2h 40m	65
HOSPITAL	0.986	2h 19m	81
RTFMP	0.987	28m	56
BPIC2012	0.791	7h 49m	126
BPIC2017	0.638	11h 15m	147

Table 6.5 presents the results of our initial studies, each conducted with 100 trials across the previously introduced event logs, using a *conditional* search space. In this setting, all samplers perform relatively similar, with `MOTPE` achieving slightly higher hypervolume scores on three of the five logs. These results suggest that, in small-scale studies with limited trials, samplers that do not model conditional dependencies efficiently can still produce competitive results.

To evaluate whether the performance of `MOTPE` improves with a larger number of trials, we conducted follow-up studies using 500 trials, again within a *conditional* search space. Given the size of our search space, which includes combinations of multiple preprocessing functions and a variety of process discovery algorithms, we expect that increasing the number of trials can help the sampler identify more promising configurations and construct a better *Pareto front*. The results are shown in Table 6.6. Although the studies were conducted for 500 trials, we report results only up to the 300th trial, as the additional gains in hypervolume beyond that point were minimal. On more complex event logs such as *SEPSIS*, *BPIC2012*, and *BPIC2017*, we observe an average hypervolume improvement of approximately 15%, with the most significant gain observed on *BPIC2017* at around 22%. These findings support the hypothesis that increasing the number of trials leads to better optimization performance, especially on large and complex event logs.

Table 6.7: Samplers performance in a flat search space (100 trials).

Event Log	**Sampler**	**Hypervolume**	**Time to Finish**	**Failed Trials**
SEPSIS	MOTPE	**0.740**	44m	17
	NSGA-III	0.580	27m	26
	NSGA-II	0.633	24m	22
	Random	0.631	52m	33
HOSPITAL	MOTPE	**0.974**	16m	18
	NSGA-III	0.925	24m	31
	NSGA-II	0.954	25m	33
	Random	0.923	22m	53
RTFMP	MOTPE	**0.978**	7m	22
	NSGA-III	0.951	14m	31
	NSGA-II	0.948	10m	21
	Random	0.924	10m	28
BPIC2012	MOTPE	**0.753**	2h 20m	30
	NSGA-III	0.686	1h 14m	53
	NSGA-II	0.704	1h 5m	50
	Random	0.697	1h 3m	52
BPIC2017	MOTPE	0.350	1h 46m	52
	NSGA-III	**0.534**	2h 30m	63
	NSGA-II	0.367	2h 35m	64
	Random	0.329	2h 50m	67

In Chapter 4, we formulated our problem as a *Combined Algorithm Selection and Hyperparameter Optimization* (CASH) problem and adopted a tree-structured conditional search space to reflect the hierarchical nature of algorithm-specific hyperparameters. To assess whether this structure leads to better optimization performance, we conducted two additional studies using *flat* search space. In these experiments, we reverted `MOTPE` to its default settings and, in the 300-trial setup, additionally included `NSGA-III` as a second sampler. The results of these studies are shown in Table 6.7 and Table 6.8. We can not directly compare the performance of samplers across *conditional* and *flat* spaces in the 100-trial studies, because the results are biased. For example, in *flat* search space, `MOTPE` improved the hypervolume on the *SEPSIS* event log, but the performance on the *BPIC2017* was worse, 0.35 compared to 0.52 in *conditional* search space. As shown in the first 300-trial studies we conducted, increasing the number of trials improves performance.

Table 6.8: Samplers performance in a flat search space (300 trials).

Event Log	Sampler	Hypervolume	Time to Finish	Failed Trials
SEPSIS	MOTPE	**0.775**	1h 40m	49
	NSGA-III	0.742	2h	61
HOSPITAL	MOTPE	**0.978**	2h 14m	54
	NSGA-III	0.975	2h 20m	58
RTFMP	MOTPE	**0.988**	30m	58
	NSGA-III	0.978	30m	54
BPIC2012	MOTPE	**0.754**	9h 48m	119
	NSGA-III	0.742	3h 30m	100
BPIC2017	MOTPE	**0.616**	12h 20m	138
	NSGA-III	0.561	6h 54m	115

From the results, we observe the following findings. First, using `MOTPE` in *conditional* search space leads to higher overall hypervolume across all event logs. However, at 300 trials, `MOTPE` consistently takes nearly twice as long as `NSGA-III` to complete studies on complex logs such as *BPIC2012* and *BPIC2017*. Interestingly, the total number of failed trials is comparable across the samplers, but the causes differ. Failures can be from several sources. Internal failures include preprocessing errors (e.g., an empty log after filtering) or exceptions thrown by process discovery algorithms. Alignment failures occur when no valid sequence can reach the final marking, indicating structural issues in the process model. Finally, timeout exceptions occur when discovery or conformance checking exceeds the predefined 5-minute threshold in any of the pipeline phases. Using `MOTPE`, the majority of failed trials on complex logs are timeouts: in the conditional setting, 50 out of 126 on *BPIC2012*, and 76 out of 147 on *BPIC2017*. A similar trend holds in the flat space configuration, 72 and 84 timeouts, respectively. In contrast, `NSGA-III` encountered significantly fewer timeouts on these logs, only 20 on *BPIC2012* and 40

on *BPIC2017*. This difference explains the increased runtime for `MOTPE`. These timeout failures suggest that `MOTPE` is selecting configurations that discover process models that require more time to align. Especially since alignments are computationally expensive, our threshold might not be enough for complex event logs. On the other hand, the lower number of timeout failures for *NSGA-III* suggests that it more often selects configurations that result in other types of failures, such as models with structural issues or preprocessing errors, indicating that its overall configuration choices tend to be less effective.

Until now, we only compared the *hypervolume* metric across different samplers and different search spaces. We saw that regardless of the search space, they perform similar, with `MOTPE` achieving a slightly higher *hypervolume* in *conditional* search space. Since `MOTPE` is the only algorithm in Optuna that fully supports both *conditional* and *flat* spaces, we do more comparisons. In Figures 6.1 and 6.2, we observe the well-known *exploration vs. exploitation* [11] trade-off, a pattern that appears in some of our studies. In hyperparameter optimization, *exploration* involves sampling diverse regions of the search space to discover promising configurations, while *exploitation* refines those already known to perform well. The figures are based on 300-trial studies over both *conditional* and *flat* search spaces, using `MOTPE`. Each dot represents the best configuration found within a specific subspace (e.g., a particular combination of preprocessing function and discovery algorithm) that has a valid result for our *multi-objective* optimization problem. The color of the dot indicates the *F1-Score* achieved by that configuration, although the Optuna Dashboard sometimes mismatches these colors, so the visual rankings may not be entirely accurate. The figues show that in a *conditional* search space, with the flags Optuna suggests, the `MOTPE` explores the search space more.

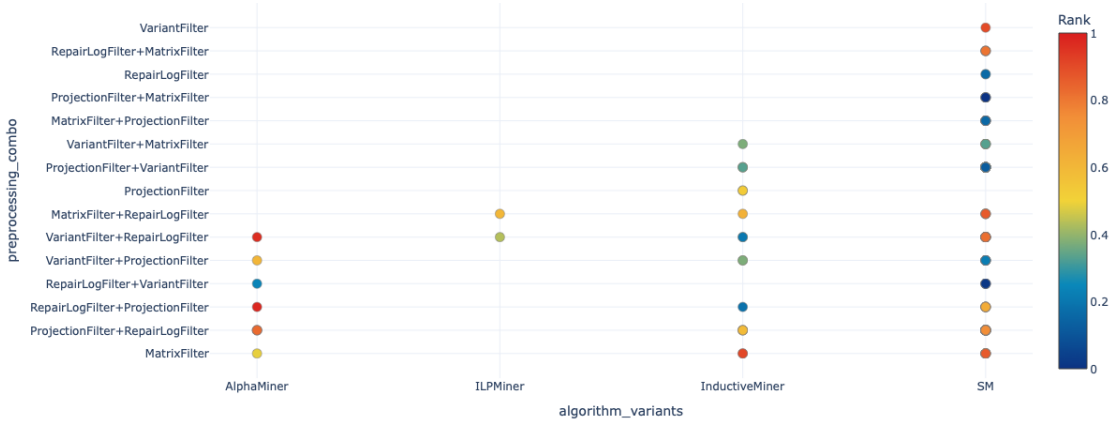


Figure 6.1: Exploration of the *flat* search space using `MOTPE` on the event log BPIC2012. The x-axis shows the discovery algorithms, and the y-axis shows whether the configuration used a one or two-step preprocessing pipeline. The dot color indicates the configuration's performance on a specific objective (F1-Score).

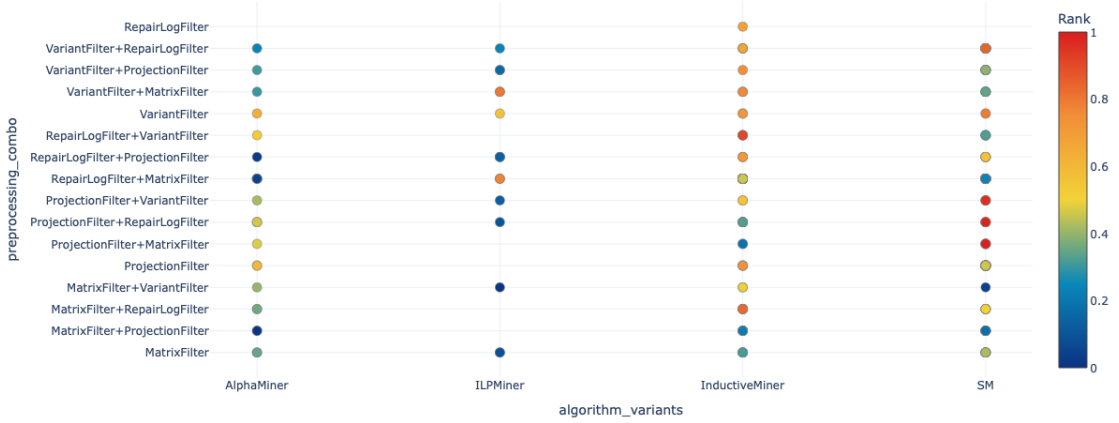


Figure 6.2: Exploration of the *conditional* search space using `MOTPE` on the event log BPIC2012.

In *conditional* spaces, the search space is decomposed into groups based on active parameters, and `MOTPE` is applied to each partitioned space to perform efficient sampling. We observed that in some studies, it leads to slower convergence initially, but allows more robust and fair exploration of the space. It is important to note that certain combinations of preprocessing and process discovery techniques may fail to yield results due to alignment issues or internal errors. Nevertheless, the difference in the number and diversity of valid configurations between the *conditional* and *flat* spaces is clearly noticeable.

Although we saw that `MOTPE` is capable of building a well-distributed *Pareto front* in *flat* search space, we found a potential problem illustrated in Figure 6.3. In *flat* search spaces, all parameters, regardless of whether they are relevant to the current configuration, are included in the optimization process. This can lead to misleading parameter importance scores, as many parameters remain inactive in most trials. Figure 6.3 shows the most influential hyperparameters for the optimization objectives, computed using fANOVA [22] (e.g., fitness, precision, F1-Score, simplicity, and generalization). The parameter `sequenceEncodingCutoffLevel` is a specific parameter used only when the discovery algorithm is `Hybrid ILP Miner`. In this study, `Hybrid ILP Miner` was sampled only 31 times from a study containing 300 trials. Each row of a parameter represents the importance for one of the quality dimensions. In this case, we see that this exact parameter has more importance than `algorithm_variants`, which in our case is the conditional decision for a process discovery algorithm. While only 31 configurations involving the `Hybrid ILP Miner` were sampled, the parameter `sequenceEncodingCutoffLevel` was still considered important. It had the same importance scores for both *fitness* and *F1-Score*. Although `Hybrid ILP Miner` ranked first in terms of *fitness*, it achieved a maximum *F1-Score* of only 0.65, compared to the best overall F1-Score of 0.82, achieved by `Split Miner`. Furthermore, since every configuration includes all possible hyperparameters, even those that are inactive, it becomes difficult to analyze the results after the study. To understand how specific parameters influence the performance of the al-

gorithms, it is necessary to know exactly which parameters are relevant to each selected algorithm. Based on those findings and also on how our problem is structured (CASH [40]), we decide to continue our evaluation, using the `MOTPE` sampler in a *conditional* space.

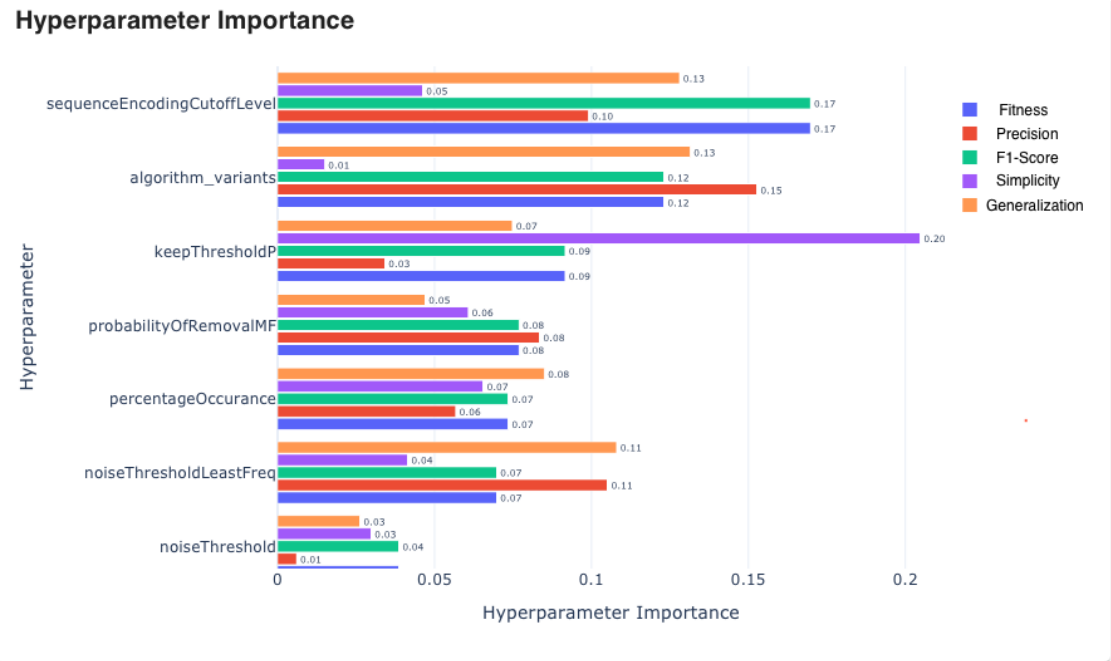


Figure 6.3: Hyperparameter importance using `MOTPE` in *flat* search space on event log BPIC2012. Result are computed using fANOVA [22]

6.4 Baseline vs. MDA

In Chapter 3, we discussed approaches similar to ours. However, direct comparison is challenging due to fundamental differences in the underlying ecosystems: our implementation is based on *ProM*, while most related work relies on *PM4Py*. This alone introduces incompatibilities in terms of available algorithms and implementation details. Furthermore, our approach integrates a broader range of process discovery techniques, whereas prior studies [14] primarily focus on the Heuristic Miner and Inductive Miner. Additionally, while related work often uses token-based replay, which may produce overly optimistic conformance results, we employ alignment-based conformance checking for a more accurate and robust evaluation. Moreover, implementation from prior work [14] is not publicly available.

Since no directly comparable baseline exists, we constructed our own. All baseline results, obtained using default parameter configurations from ProM, are presented in Appendix 7. To ensure that the results are comparable, every default configuration was executed through the pipeline we implemented. We considered three default configura-

tion setups for comparison. First, each process discovery algorithm was applied directly to the raw event log, without preprocessing. Next, we introduced a single preprocessing step using the *Variant Filter* (`1PP`). Finally, we tested a two-step preprocessing pipeline (`2PP`), applying the *Project Traces* followed by the *Variant Filter* (other direction might remove events that would not be removed otherwise). For both preprocessing configurations, thresholds were set to 80% following the well-known 80/20 principle [2], retaining the most frequent behavior.

There are many valid ways to evaluate the quality of a process model, and the appropriate perspective often depends on the intended use case. For example, a compliance auditor might prioritize fitness and precision, while a business analyst may value simplicity and interpretability. As such, rather than relying on a single metric, we present results from several quality dimensions.

We begin by evaluating fitness in isolation to ensure that the model captures the observed behavior in the event log. Fitness is a necessary condition for any useful process model [13].

We then assess the combination of fitness and simplicity to examine whether the observed behavior can be captured without introducing unnecessary structural complexity. This is particularly important when decisions must be communicated to stakeholders without a technical background.

A process model with high generalization is essential for implementing a process in a business context. Such a model should capture not only the observed activity sequences from the event log but also be capable of allowing unseen, future behavior. Thus, we examine fitness and generalization together.

In addition, we examine precision and generalization together, restricting to models with fitness of at least 0.8. This allows us to focus on process models that provide a balance between *underfitting* and *overfitting*.

To capture the balance between fitness and precision, we use the F1-score, the harmonic mean of the two. This score is particularly relevant in contexts where the process model should capture the underlying process very accurately.

Finally, we present a weighted score, assigning equal weight to all four metrics: *fitness, precision, generalization, and simplicity*. This offers a balanced evaluation of model quality when no single metric is favored.

In scenarios where two or more objectives are considered, we assume no preference between them. Instead, we sum their respective scores and compare the overall result across all configurations. This comparison is performed for both the `default` configurations and those generated by `MDA`. The `MDA` configurations included in the comparison are *Pareto-optimal*, representing non-dominated solutions. As a result, our choices from `MDA` and `default` configurations reflect trade-offs: a configuration with higher simplicity but lower fitness could still be preferred if the overall combined score remains competitive, and also the other way around.

We include these different evaluation perspectives to reflect a variety of real-life use cases. This allows us to demonstrate that our algorithm consistently discovers process models with high values in all quality metrics, regardless of the specific priorities. These

results support our claim that combining preprocessing, process discovery, and hyperparameter tuning leads to better process models across diverse quality dimensions.

In the following `MF` refers to Matrix Filter, `VF` to Variant Filter and `PJ` to Projection Filter. Inductive Miner to `IM`, with its variant in the brackets, Split Miner to `SM`, and Hybrid ILP Miner to `HILPM`.

Table 6.9: Comparison of default and MDA-optimized configurations by Fitness.

Event Log	**Default**	**Fitness**	**MDA**	**Fitness**
SEPSIS	IM(IMf)_1PP	0.914	IM(IM)_MF+PF	**1.00**
HOSPITAL	IM(IMf)	0.921	SM_VF	**1.00**
RTFMP	IM(IMf)	0.982	IM(IMc)_PF	**1.00**
BPIC2012	SM_2PP	0.842	IM(IM)_MF+PF	**0.945**
BPIC2017	SM_2PP	0.385	IM(IMc)_RL+MF	**0.829**

Table 6.10: Comparison of default and MDA-optimized configurations by Fitness+Simplicity.

Event Log	**Default**	**Fitness/Simplicity**	**MDA**	**Fitness/Simplicity**
SEPSIS	IM(IMf)_1PP	0.914/0.969	IM(IM)_MF+PF	**1.00/1.00**
HOSPITAL	IM(IMf)	0.921/**1.00**	SM_VF	**1.00/1.00**
RTFMP	IM(IMf)	0.982/**1.00**	IM(IM)_VF	**1.00/1.00**
BPIC2012	SM_2PP	0.842/0.914	IM(IMa)_MF	**0.936/0.965**
BPIC2017	SM_2PP	0.385/0.827	IM(IMc)_RL+MF	**0.829/0.869**

Table 6.11: Comparison of default and MDA-optimized configurations by Fitness+Generalization.

Event Log	**Default**	**Fitness/Generalization**	**MDA**	**Fitness/Generalization**
SEPSIS	IM(IMf)_1PP	0.914/0.935	IM(IMc)_RL+VF	**0.991/0.945**
HOSPITAL	IM(IMf)_1PP	0.918/0.955	IM(IMc)_MF+VF	**0.987/0.991**
RTFMP	IM(IMf)	0.982/0.944	IM(IMlc)_RL+PF	**0.998/0.994**
BPIC2012	SM_2PP	0.842/0.987	IM(IM)_MF+PF	**0.945/0.988**
BPIC2017	SM_2PP	0.385/0.972	IM(IMc)_RL+MF	**0.825/0.992**

Table 6.12: Comparison of default and MDA-optimized configurations by Precision+Generalization. Fitness must be at least 0.8

Event Log	Default	Precision/Generalization	MDA	Precision/Generalization
SEPSIS	IM_2PP	0.597/0.970	SM_VF	**0.917/0.949**
HOSPITAL	SM	0.966/0.966	SM_VF	**1.00/0.996**
RTFMP	HILPM	**1.00/0.977**	HILPM_MF+VF	**1.00/0.997**
BPIC2012	SM_2PP	0.910/0.987	SM_RL+MF	**0.966/0.984**
BPIC2017	-	-	IM(IMf)_RL+MF	**0.778/0.994**

Table 6.13: Comparison of default and MDA-optimized configurations by F1-Score.

Event Log	Default	F1-score	MDA	F1-score
SEPSIS	HILPM_2PP	0.7869	SM_VF	**0.8657**
HOSPITAL	IM_1PP	0.9573	SM_VF	**0.9923**
RTFMP	SM	0.9383	SM_RL+MF	**0.9926**
BPIC2012	SM_2PP	0.8755	SM_RL+MF	**0.911**
BPIC2017	SM_2PP	0.5550	SM_RL+MF	**0.8365**

Table 6.14: Comparison of default and MDA-optimized configurations by weighted score

Event Log	Default	Weighted Score	MDA	Weighted Score
SEPSIS	HILPM_2PP	0.8580	SM_RL+VF	**0.9190**
HOSPITAL	SM	0.9342	SM_VF	**0.9855**
RTFMP	SM	0.9609	SM_PF	**0.9915**
BPIC2012	SM_2PP	0.9137	SM_RL+MF	**0.9446**
BPIC2017	SM_2PP	0.7934	SM_RL+MF	**0.8951**

Tables 6.9 through 6.14 present a comparison between `MDA` and `default` configurations. The results for `MDA` were obtained using 500-trial studies with the `MOTPE` sampler in a *conditional* search space. Across all evaluation scenarios `MDA` configurations consistently achieve higher results in the quality dimensions compared to the `default` configuration. The baseline columns include the algorithms that performed best in the evaluated scenarios. Before we analyze the results, two important things must be considered. *ProM's* default choice of Inductive Miner variant is `infrequent` [28], which does not guarantee perfect *fitness*. We also observed in 6.3 that, depending on the event log, some studies can take several hours to complete. Although improvements were achieved, they may have come at a relatively high cost in terms of time.

A key observation from the baseline results is the significant impact of preprocessing. In most cases, applying preprocessing before process discovery led to improved outcomes. This highlights that selecting an appropriate discovery algorithm alone is often insufficient for achieving optimal results. However, preprocessing does not guar-

antee improvement in every scenario. For instance, on event logs such as *RTMFP* and *HOSPITAL*, preprocessing did not result in better process models. However, on more complex event logs with many unique traces like *BPIC2017, BPIC2012, SEPSIS*, the *sequential* application of preprocessing techniques proved efficient. Given the vast number of hyperparameter combinations, it is challenging to precisely quantify the extent of behavioral change introduced by each preprocessing step. Investigating which combinations and parameter settings yield consistently strong results remains a promising direction for future work.

Interesting and to some extent expected, although our pipeline includes five distinct process discovery algorithms: Inductive Miner, Split Miner, Hybrid ILP Miner, Alpha Miner, and Heuristics Miner, and their variants, only three consistently appear in the top-performing `MDA` configurations: Split Miner (SM), Hybrid ILP Miner (HILPM), Inductive Miner (IM) with its variants. These algorithms frequently contribute to the configurations representing the *Pareto front*. When fitness is prioritized, Inductive Miner variants (e.g., IM, IMlc) are frequently selected due to their formal guarantees. However, these process models often come at the cost of lower precision. In contrast, Split Miner is frequently chosen in multi-objective contexts, particularly where precision and fitness are important. Although Hybrid ILP Miner theoretically offers both high fitness and precision, it rarely appears in the top-performing `MDA` configurations. Despite including the variants of the Alpha Miner that address some of the issues with the original Alpha Miner algorithm, `MDA` did not manage to find configurations that discover a process model suitable for one of the scenarios we presented. Even though benchmarks [43] suggest that the Heuristics Miner is well-suited for real-life data, we do not see it across the *Pareto-optimal* configurations.

In summary, the results provide strong evidence for the benefits of hyperparameter tuning and preprocessing in process discovery. The `MDA` approach consistently improves model quality across multiple scenarios, underscoring the importance of both tuning and careful preprocessing in discovery pipelines. Although we did not observe substantial improvements on every event log for each evaluated scenario, notable gains were observed in several cases. For instance, as shown in Table 6.13, the `MDA` configuration results in an *F1-score* that is around 50% better than the one obtained by the default configuration. To further illustrate this, we compare process models discovered using default configurations and those discovered by `MDA`. We present a representative case, where `MDA` yields a significant improvement.

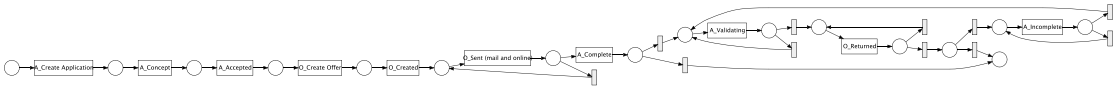


Figure 6.4: Best F1-Score obtained on the event log BPIC2017 from a default configuration

In Figure 6.4 and Figure 6.5, we show the process models discovered using the `default` and `MDA` configurations, respectively. The model from the `default` configuration achieves

a high *precision* of 0.98 but a very low *fitness* of 0.385, resulting in an overall *F1-score* of only 0.55. This indicates an *oversimplified* process model that fails to replay much of the behavior recorded in the event log. In contrast, the model discovered using `MDA` strikes a better balance, with a *fitness* of 0.76 and a *precision* of 0.92, leading to an *F1-score* of 0.836. The `MDA` model allows more behavior while maintaining high precision, resulting in a more accurate representation of the underlying process.

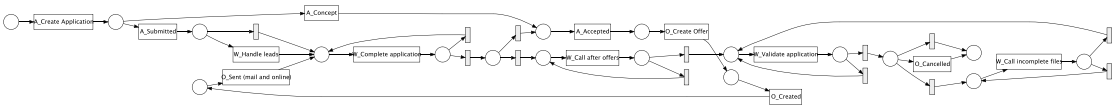


Figure 6.5: Best F1-Score obtained on the event log BPIC2017 from `MDA` configuration

6.5 Effect of Pruning and Preprocessing

We turn to the third main objective of our evaluation: *Identifying potential improvements in our configuration.* In this part, we aim to answer the following questions: How does pruning affect the sampler? How important is preprocessing?

6.5.1 Pruning vs. Zero Quality Scores

We begin by exploring the role of pruning in our current setup. As previously discussed, pruning in a *multi-objective* setting aims to stop unpromising trials early, ideally improving optimization efficiency. However, our current implementation does not provide meaningful feedback during pruning. To better understand the effect of the lack of feedback, we modify the pruning behavior. Instead of pruning, we return zero quality scores whenever a trial would have been pruned. This allows us to simulate how the sampler responds to ”bad” feedback. While it is possible to return more informative values that better reflect the estimated process model quality, we leave the exploration of such strategies for future work. By doing this modification, we expect to particularly affect event logs where pruning is triggered more frequently, potentially influencing the overall optimization trajectory.

In the following Tables 6.15 and 6.16, ZQS denotes zero quality scores.

The results of our 100 and 300-trial studies are presented in Tables 6.15, 6.16. We observe that across all event logs there is a difference in the *hypervolume* achieved. Our initial evaluation showed that the sampler performs better when more trials are allowed. In the 100-trial studies, for event logs such as *SEPSIS* and *BPIC2017*, pruning led to slightly better configurations. The effect becomes more pronounced in the 300-trial studies, particularly for *BPIC2012* and *BPIC2017*, where the difference in hypervolume is more substantial. For example, on *BPIC2017*, pruning results in a hypervolume approximately 20% larger than the zero quality approach, indicating a noticeable improvement in the overall quality of the discovered *Pareto front.* This is also reflected in the achieved *F1-scores.*

Table 6.15: Pruning or ZQS?

Event Log	Pruning (100)	ZQS (100)	Pruning (300)	ZQS (300)
SEPSIS	**0.704**	0.672	**0.781**	0.780
HOSPITAL	**0.968**	0.967	**0.986**	0.978
RTFMP	**0.985**	0.974	**0.987**	**0.987**
BPIC2012	0.701	**0.710**	**0.791**	0.748
BPIC2017	**0.524**	0.505	**0.638**	0.533

Table 6.16: Achieved *F1-Scores* with pruning and ZQS

Event Log	Pruning (100)	ZQS (100)	Pruning (300)	ZQS (300)
SEPSIS	0.826	**0.834**	**0.865**	0.862
HOSPITAL	0.987	**0.993**	**0.992**	0.989
RTFMP	**0.995**	0.987	0.992	**0.996**
BPIC2012	0.880	**0.887**	**0.911**	0.894
BPIC2017	**0.785**	0.693	**0.836**	0.786

In the studies conducted so far, we observe that even without proper feedback, when pruning, the sampler manages to find configurations that improve the *Pareto front*. However, this comes with significant inefficiency. As presented in Table 6.6, for the *BPIC2017* log, 147 out of 300 trials were pruned, meaning 49% of the sampled configurations did not return valid results. In theory, providing feedback should increase the efficiency of the sampler. However, we must be careful what we return. Figure 6.6 illustrates the *hypervolume* progression over trials. The blue line represents the study with pruning on the *BPIC2017* event log, while the red line corresponds to a study using zero quality results. Up to the 40th trial, the zero quality approach achieves a higher hypervolume. This early advantage may be attributed to Optuna's `MOTPE` implementation, which relies on random sampling during the initial trials. However, the final *hypervolume* after 300 trials is similar to the one achieved by pruning after just 50 trials. This highlights the idea that misleading feedback can degrade the optimization performance. Even a small change in the feedback might prevent further exploration of subspaces containing configurations that could result in high-quality process models.

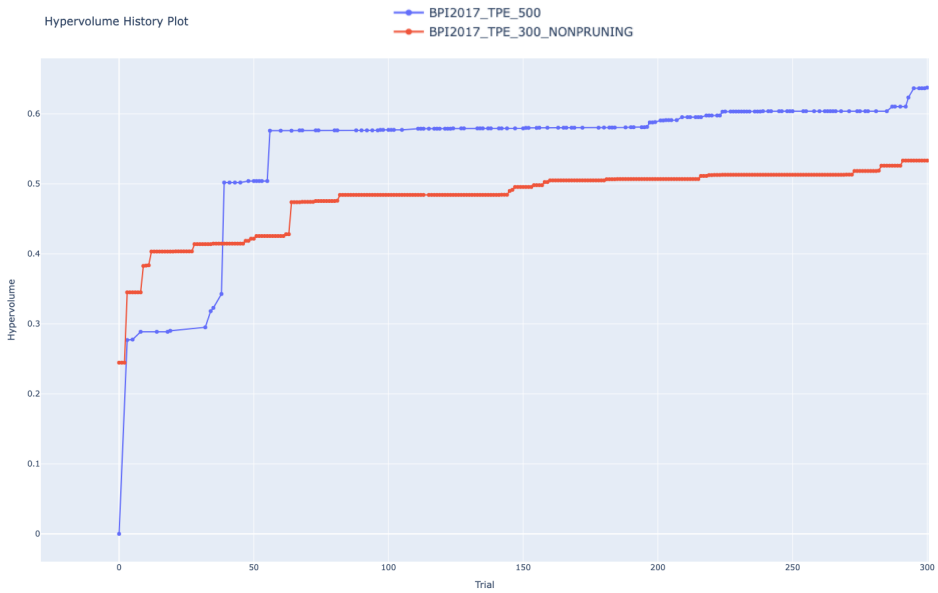
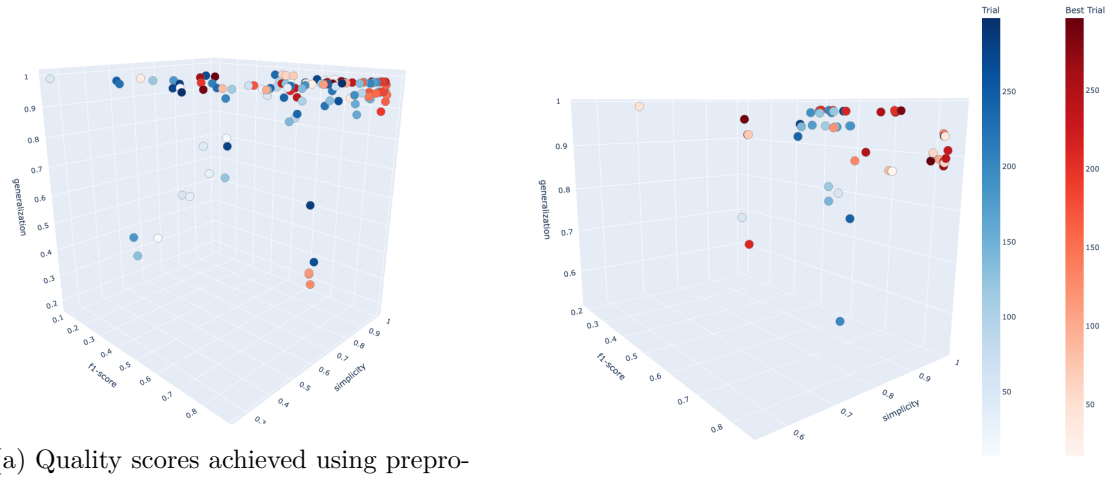


Figure 6.6: Effect of zero quality results on the hypervolume progression over 300 trials on the event log BPIC2017

6.5.2 Effect of preprocessing

Throughout this thesis, we put a lot of emphasis on preprocessing. The baseline comparison showed that, across the different scenarios, the configurations that achieved the best results for the targeted quality metrics consistently involved some form of *preprocessing*. To see to what extend *preprocessing* contributes to the *Pareto front*, we conducted a 300-trial studies using the *BPIC2012* and *SEPSIS* logs. These two logs were selected for their structural complexity and variability, allowing us to compare the resulting *Pareto fronts* with and without preprocessing, and to assess whether tuning discovery algorithm hyperparameters alone is sufficient to discover high-quality configurations. In Figures 6.7a, 6.7b we see notable results. Both figures represent the *Pareto front* of the studies in three dimensions, conducted on the *BPIC2012* event log. We used *generalization*, *F1-score*, and *simplicity*. The *Pareto-optimal* configurations are marked with red dots. The widespread along the F1-score axis in Figure 6.7a indicates that preprocessing helps the sampler to discover diverse process models, each reflecting different trade-offs between fitness and precision. The clusters near (1,1,1) suggest that some configurations achieve a near-optimal balance across all three quality dimensions.

In Figure 6.7b we see different behavior. Overall, we see that fewer points are scattered across the axes. Without preprocessing, only 93 trials have been pruned, compared to 126. While fewer pruned trials might suggest more diverse results in theory, the opposite is observed. The resulting configurations show limited diversity, with many configurations clustering around similar quality scores. These findings suggest that tuning process discovery algorithms alone can still produce configurations with high-quality values in



(a) Quality scores achieved using preprocessing.

(b) Quality scores achieved without preprocessing.

Figure 6.7: This figure shows 3D *Pareto front* distributions on the BPIC2012 event log from 300-trial studies, comparing configurations with and without preprocessing. Red points represent *Pareto-optimal* configurations, highlighting the trade-offs among *F1-Score*, *simplicity*, and *generalization*.

the different quality dimensions, but without preprocessing, the range of trade-offs explored remains limited.

6.6 Discussion

Our evaluation was structured around three main objectives. First, we assessed the behavior of different samplers in both flat and conditional search spaces. The results showed that `MOTPE` performs best in our conditional setting, achieving overall higher hypervolume scores. However, this comes at the cost of increased runtime, particularly on complex event logs. We also discussed limitations of *flat* search spaces, such as unbalanced *exploration* vs. *exploitation* and misleading hyperparameter importance scores. Second, we demonstrated that `MDA` configurations consistently outperform `default` settings by producing process models with higher values across all quality dimensions. These improvements, however, often require hundreds of trials. Third, we analyzed the impact of pruning by comparing it to returning zero quality scores. The results suggest that pruning is effective in our current setup, but more informative feedback could improve the efficiency of the sampler. Finally, we highlighted the importance of preprocessing. Including preprocessing, the resulting configurations form a more diverse *Pareto front*, capturing different trade-offs between the quality dimensions considered.

6.7 Threats to Validity

We designed our evaluation around real-life event logs, which tend to be less structured than synthetic or artificially cleaned event logs. Although the selected logs vary significantly, from relatively simple to highly complex, involving millions of events, they still represent a limited sample of five logs. Due to the substantial computational cost, each study was conducted only once. In several cases, a single 500-trial study took over 12 hours to complete, which limited our ability to repeat experiments to validate the achieved results. Finally, in some experiments, especially in the last section of our evaluation, we used only 2 or 3 event logs, which might not be enough to fully confirm our findings.

7 Conclusion

Process mining enables the analysis and improvement of business processes by extracting insights from event data. However, fully leveraging process discovery presents several challenges. First, users must select an appropriate discovery algorithm based on their specific goals, a task that is both time-consuming and error-prone, even with domain knowledge of quality metric trade-offs each algorithm presents. Second, real-life event data often contains noise, outliers, and infrequent behavior, which can lead to overly complex or inaccurate models. This necessitates effective preprocessing, however, it also requires careful algorithm selection and tuning based on specific goals.

To address these challenges, we introduced the *Meta-Discovery Algorithm*, a multi-objective optimization framework that jointly selects and tunes both preprocessing and process discovery algorithms. `MDA` is formulated as a *Combined Algorithm Selection and Hyperparameter* optimization problem, where each configuration, comprising preprocessing steps, process discovery method, and their respective hyperparameters, is evaluated across multiple quality dimensions: *fitness, precision, simplicity*, and *generalization*. This formulation enables the exploration of trade-offs among conflicting objectives and yields a *Pareto front* of diverse, high-quality process models.

Future Work

In our evaluation, we covered some directions of possible further improvements of our approach.

- **Extending the Search Space**: `MDA` support a fixed set of discovery and preprocessing techniques. Extending the search space to include more techniques could further improve our results.
- **Optimization Strategy and Focus**: `MDA` currently relies on the default samplers provided by Optuna. Future work could explore the development of custom sampling strategies that are better suited to the hierarchical and conditional nature of our process discovery pipeline. As discussed in Chapter 6, certain challenges, such as effective pruning, can arise in this context. Designing meaningful intermediate feedback values (e.g., partial quality metrics) could improve the guidance provided to the sampler during optimization and significantly enhance overall efficiency.
- **Algorithm Recommendation**: `MDA` can benefit from Meta-learning approaches [24]. Instead of starting each optimization from scratch, meta-learning can leverage prior knowledge from previously analyzed event logs to predict a suitable combination of preprocessing and process discovery algorithms for new event logs. This

can significantly reduce the search space and mitigate the potential curse of dimensionality, especially as more algorithms and their respective hyperparameters are added to the search space. As a result, `MDA` would be able to converge more quickly to high-quality models, without compromising the diversity in the *Pareto front*.

A Evaluation Results Default Settings

Table A.1: Default configuration results on the **SEPSIS** event log (including failed trials).

Algorithm	**Fitness**	**Precision**	**F1-Score**	**Simplicity**	**Generalization**
AlphaMiner	0.1734	0.4603	0.2519	1.0000	0.2304
AlphaMiner_1PP	0.1691	0.3750	0.2331	1.0000	0.1737
AlphaMiner_2PP	0.2339	0.7952	0.3615	0.6400	0.7641
InductiveMiner	0.9030	0.5353	0.6721	0.9683	0.9357
InductiveMiner_1PP	0.9140	0.5298	0.6708	0.9697	0.9353
InductiveMiner_2PP	0.8206	0.5973	0.6914	0.7907	0.9700
Split Miner	–	–	–	–	–
Split Miner_1PP	–	–	–	–	–
Split Miner_2PP	–	–	–	–	–
HybridILP	0.6774	0.7750	0.7229	0.9286	0.8960
HybridILP_1PP	0.6887	0.7721	0.7280	0.9306	0.8960
HybridILP_2PP	0.6487	1.0000	0.7869	0.8125	0.9711
HeuristicsMiner	0.4822	0.7651	0.5916	1.0000	0.7827
HeuristicsMiner_1PP	0.4822	0.7651	0.5916	1.0000	0.8001
HeuristicsMiner_2PP	–	–	–	–	–

Table A.2: Default configuration results on the **HOSPITAL** event log (including failed trials).

Algorithm	Fitness	Precision	F1-Score	Simplicity	Generalization
AlphaMiner	0.4913	0.2710	0.3493	1.0000	0.2058
AlphaMiner_1PP	–	–	–	–	–
AlphaMiner_2PP	0.7211	1.0000	0.8379	0.5517	0.9968
InductiveMiner	0.9218	0.7851	0.8480	1.0000	0.9101
InductiveMiner_1PP	0.9181	1.0000	0.9573	0.6364	0.9956
InductiveMiner_2PP	0.7211	1.0000	0.8379	0.5517	0.9968
Split Miner	0.8058	0.9651	0.8783	1.0000	0.9663
Split Miner_1PP	0.7235	1.0000	0.8396	0.6571	0.9968
Split Miner_2PP	0.7211	1.0000	0.8379	0.6061	0.9968
HybridILP	0.7235	1.0000	0.8396	0.7333	0.9968
HybridILP_1PP	0.7235	1.0000	0.8396	0.6757	0.9968
HybridILP_2PP	0.7211	1.0000	0.8379	0.6061	0.9968
HeuristicsMiner	–	–	–	–	–
HeuristicsMiner_1PP	0.7590	1.0000	0.8630	0.7143	0.9963
HeuristicsMiner_2PP	0.7211	1.0000	0.8379	0.6486	0.9968

Table A.3: Default configuration results on the **Road Traffic Management** event log (including failed trials).

Algorithm	Fitness	Precision	F1-Score	Simplicity	Generalization
AlphaMiner_2PP	0.8394	1.0000	0.9127	0.7143	0.9966
InductiveMiner	0.9823	0.8517	0.9124	1.0000	0.9441
InductiveMiner_1PP	0.8769	1.0000	0.9344	0.7826	0.9966
InductiveMiner_2PP	0.8394	1.0000	0.9127	0.7143	0.9966
Split Miner	0.8838	1.0000	0.9383	1.0000	0.9601
Split Miner_1PP	0.8769	1.0000	0.9344	0.8718	0.9968
Split Miner_2PP	0.8394	1.0000	0.9127	0.8378	0.9968
HybridILP	0.6731	1.0000	0.8046	0.7778	0.9974
HybridILP_1PP	0.8769	1.0000	0.9344	0.8148	0.9968
HybridILP_2PP	0.8394	1.0000	0.9127	0.7600	0.9968
HeuristicsMiner	–	–	–	–	–
HeuristicsMiner_1PP	0.5783	0.7500	0.6531	0.8529	0.2720
HeuristicsMiner_2PP	0.6698	0.8750	0.7588	0.8000	0.7758

Table A.4: Default configuration results on the **BPIC2012** event log (including failed trials).

Algorithm	**Fitness**	**Precision**	**F1-Score**	**Simplicity**	**Generalization**
AlphaMiner	–	–	–	–	–
AlphaMiner_1PP	–	–	–	–	–
AlphaMiner_2PP	0.5634	0.2694	0.3645	0.7142	0.9009
InductiveMiner	–	–	–	–	–
InductiveMiner_1PP	–	–	–	–	–
InductiveMiner_2PP	0.7266	0.9284	0.8152	0.8356	0.9810
Split Miner	–	–	–	–	–
Split Miner_1PP	–	–	–	–	–
Split Miner_2PP	0.8426	0.9110	0.8755	0.9142	0.9870
HybridILP	0.5674	0.5778	0.5726	0.6222	0.8817
HybridILP_1PP	0.7233	0.5837	0.6546	0.6222	0.9907
HybridILP_2PP	0.4742	0.8333	0.6044	0.5000	0.7084
HeuristicsMiner	–	–	–	–	–
HeuristicsMiner_1PP	–	–	–	–	–
HeuristicsMiner_2PP	–	–	–	–	–

Table A.5: Default configuration results on the **BPIC2017** event log (including failed trials).

Algorithm	**Fitness**	**Precision**	**F1-Score**	**Simplicity**	**Generalization**
AlphaMiner	–	–	–	–	–
AlphaMiner_1PP	–	–	–	–	–
AlphaMiner_2PP	0.2663	0.2498	0.257	0.7333	0.7656
InductiveMiner	–	–	–	–	–
InductiveMiner_1PP	–	–	–	–	–
InductiveMiner_2PP	–	–	–	–	–
Split Miner	–	–	–	–	–
Split Miner_1PP	–	–	–	–	–
Split Miner_2PP	0.3859	0.9879	0.5550	0.8279	0.9721
HybridILP	–	–	–	–	–
HybridILP_1PP	–	–	–	–	–
HybridILP_2PP	–	–	–	–	–
HeuristicsMiner	–	–	–	–	–
HeuristicsMiner_1PP	–	–	–	–	–
HeuristicsMiner_2PP	–	–	–	–	–

Bibliography

[1] W. M. P. van der Aalst. *Process Mining: Data Science in Action, Second Edition.* Springer, 2016, pp. 125–138. ISBN: 978-3-662-49851-4. DOI: 10.1007/978-3-662-49851-4.

[2] W. M. P. van der Aalst. "On the Pareto Principle in Process Mining, Task Mining, and Robotic Process Automation." In: *Proceedings of the 9th International Conference on Data Science, Technology and Applications, DATA 2020, Lieusaint, Paris, France, July 7-9, 2020.* Ed. by S. Hammoudi, C. Quix, and J. Bernardino. SciTePress, 2020, pp. 5–12.

[3] W. M. P. van der Aalst. "Foundations of Process Discovery." In: *Process Mining Handbook.* Ed. by W. M. P. van der Aalst and J. Carmona. Vol. 448. Lecture Notes in Business Information Processing. Springer, 2022, pp. 37–75. DOI: 10.1007/978-3-031-08848-3_2.

[4] W. M. P. van der Aalst, A. Adriansyah, and B. van Dongen. "Replaying history on process models for conformance checking and performance analysis." In: *WIREs Data Mining and Knowledge Discovery* 2.2 (2012), pp. 182–192. DOI: 10.1002/widm.1045.

[5] W. M. P. van der Aalst, T. Weijters, and L. Maruster. "Workflow Mining: Discovering Process Models from Event Logs." In: *IEEE Trans. Knowl. Data Eng.* 16.9 (2004), pp. 1128–1142. DOI: 10.1109/TKDE.2004.47.

[6] S. F. Adra, T. J. Dodd, I. Griffin, and P. J. Fleming. "Convergence Acceleration Operator for Multiobjective Optimization." In: *IEEE Trans. Evol. Comput.* 13.4 (2009), pp. 825–847. DOI: 10.1109/TEVC.2008.2011743.

[7] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. "Optuna: A Next-generation Hyperparameter Optimization Framework." In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019.* Ed. by A. Teredesai, V. Kumar, Y. Li, R. Rosales, E. Terzi, and G. Karypis. ACM, 2019, pp. 2623–2631. DOI: 10.1145/3292500.3330701.

[8] A. Augusto, R. Conforti, M. Dumas, M. L. Rosa, F. M. Maggi, A. Marrella, M. Mecella, and A. Soo. "Automated Discovery of Process Models from Event Logs: Review and Benchmark." In: *IEEE Trans. Knowl. Data Eng.* 31.4 (2019), pp. 686–705. DOI: 10.1109/TKDE.2018.2841877.

[9] A. Augusto, R. Conforti, M. Dumas, M. L. Rosa, and A. Polyvyanyy. "Split miner: automated discovery of accurate and simple business process models from event logs." In: *Knowl. Inf. Syst.* 59.2 (2019), pp. 251–284. DOI: 10.1007/S10115-018-1214-X.

[10] A. Berti, S. J. van Zelst, and D. Schuster. "PM4Py: A process mining library for Python." In: *Softw. Impacts* 17 (2023), p. 100556. DOI: 10.1016/J.SIMPA.2023.100556.

[11] A. Bondu, V. Lemaire, and M. Boullé. "Exploration vs. exploitation in active learning : A Bayesian approach." In: *International Joint Conference on Neural Networks, IJCNN 2010, Barcelona, Spain, 18-23 July, 2010*. IEEE, 2010, pp. 1–7. DOI: 10.1109/IJCNN.2010.5596815.

[12] R. P. J. C. Bose and W. M. P. van der Aalst. "Trace Alignment in Process Mining: Opportunities for Process Diagnostics." In: *Business Process Management - 8th International Conference, BPM 2010, Hoboken, NJ, USA, September 13-16, 2010. Proceedings*. Ed. by R. Hull, J. Mendling, and S. Tai. Vol. 6336. Lecture Notes in Computer Science. Springer, 2010, pp. 227–242. DOI: 10.1007/978-3-642-15618-2_17.

[13] J. C. A. M. Buijs, B. F. van Dongen, and W. M. P. van der Aalst. "Quality Dimensions in Process Discovery: The Importance of Fitness, Precision, Generalization and Simplicity." In: *Int. J. Cooperative Inf. Syst.* 23.1 (2014). DOI: 10.1142/S0218843014400012.

[14] S. P. C. Cardoso. "Optimizing Process Mining Algorithms: A Hyperparameter Tuning Approach." In: *CoRR* (2023). URL: https://hdl.handle.net/10216/156978.

[15] J. Carmona, B. F. van Dongen, and M. Weidlich. "Conformance Checking: Foundations, Milestones and Challenges." In: *Process Mining Handbook*. Ed. by W. M. P. van der Aalst and J. Carmona. Vol. 448. Lecture Notes in Business Information Processing. Springer, 2022, pp. 155–190. DOI: 10.1007/978-3-031-08848-3_5.

[16] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. "A fast and elitist multiobjective genetic algorithm: NSGA-II." In: *IEEE Transactions on Evolutionary Computation* 6.2 (2002), pp. 182–197. DOI: 10.1109/4235.996017.

[17] K. Deb and H. Jain. "An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints." In: *IEEE Transactions on Evolutionary Computation* 18.4 (2014), pp. 577–601. DOI: 10.1109/TEVC.2013.2281535.

[18] B. van Dongen. *BPI Challenge 2012*. 2012. DOI: 10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f.

[19] B. van Dongen. *BPI Challenge 2017*. 2017. DOI: 10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b.

[20] M. Feurer and F. Hutter. "Hyperparameter Optimization." In: *Automated Machine Learning - Methods, Systems, Challenges*. Ed. by F. Hutter, L. Kotthoff, and J. Vanschoren. The Springer Series on Challenges in Machine Learning. Springer, 2019, pp. 3–33. DOI: 10.1007/978-3-030-05318-5_1.

[21] G. V. Houdt, M. de Leoni, N. Martin, and B. Depaire. "An empirical evaluation of unsupervised event log abstraction techniques in process mining." In: *Inf. Syst.* 121 (2024), p. 102320. DOI: 10.1016/J.IS.2023.102320.

[22] F. Hutter, H. H. Hoos, and K. Leyton-Brown. "An Efficient Approach for Assessing Hyperparameter Importance." In: *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*. Vol. 32. JMLR Workshop and Conference Proceedings. JMLR.org, 2014, pp. 754–762. URL: http://proceedings.mlr.press/v32/hutter14.html.

[23] H. Jain and K. Deb. "An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point Based Nondominated Sorting Approach, Part II: Handling Constraints and Extending to an Adaptive Approach." In: *IEEE Transactions on Evolutionary Computation* 18.4 (2014), pp. 602–622. DOI: 10.1109/TEVC.2013.2281534.

[24] S. B. Jr., P. Ceravolo, E. Damiani, and G. M. Tavares. "Using Meta-learning to Recommend Process Discovery Methods." In: *CoRR* abs/2103.12874 (2021). arXiv: 2103.12874. URL: https://arxiv.org/abs/2103.12874.

[25] F. Karl, T. Pielok, J. Moosbauer, F. Pfisterer, S. Coors, M. Binder, L. Schneider, J. Thomas, J. Richter, M. Lang, E. C. Garrido-Merchán, J. Branke, and B. Bischl. "Multi-Objective Hyperparameter Optimization in Machine Learning - An Overview." In: *ACM Trans. Evol. Learn. Optim.* 3.4 (2023), 16:1–16:50. DOI: 10.1145/3610536.

[26] S. J. J. Leemans. "Robust Process Mining with Guarantees." PhD thesis. Eindhoven, The Netherlands: Eindhoven University of Technology, 2017. URL: https://pure.tue.nl/ws/portalfiles/portal/88310804/20170628_LEEMANS.pdf.

[27] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst. "Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour." In: *Business Process Management Workshops*. Vol. 171. Lecture Notes in Business Information Processing. Springer, 2014, pp. 66–78. DOI: 10.1007/978-3-319-06257-0_6.

[28] S. J. Leemans, D. Fahland, and W. M. van der Aalst. "Discovering block-structured process models from event logs." In: *International Conference on Applications and Theory of Petri Nets*. Springer. 2013, pp. 319–336.

[29] M. (de Leoni and F. Mannhardt. *Road Traffic Fine Management Process*. 2015. DOI: 10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5.

[30] J.-C. Levesque, F. Durand, and C. Gagné. "Bayesian Optimization for Conditional Hyperparameter Spaces." In: *International Joint Conference on Neural Networks (IJCNN)*. 2017, pp. 286–293. DOI: 10.1109/IJCNN.2017.7965868.

[31] X. Li, A. Makarova, and F. Hutter. "Modeling All Response Surfaces in One for Conditional Search Spaces." In: *arXiv preprint arXiv:2501.04260* (2025).

[32] F. Mannhardt. *Sepsis Cases - Event Log*. 2016. DOI: 10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460.

[33] F. Mannhardt. *Hospital Billing - Event Log*. 2017. DOI: 10.4121/uuid:76c46b83-c930-4798-a1c9-4be94dfeb741.

[34] H. Marin-Castro and E. Tello. "Event Log Preprocessing for Process Mining: A Review." In: *Applied Sciences* 11 (Nov. 2021), p. 10556. DOI: 10.3390/app112210556.

[35] A. A. de Medeiros, B. van Dongen, W. M. P. van der Aalst, and A. J. M. M. Weijters. *Process Mining: Extending the -Algorithm to Mine Short Loops*. Technical Report WP 113. Eindhoven University of Technology, 2004.

[36] Y. Ozaki, Y. Tanigaki, S. Watanabe, and M. Onishi. "Multiobjective tree-structured parzen estimator for computationally expensive optimization problems." In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. GECCO '20. Cancún, Mexico: Association for Computing Machinery, 2020, pp. 533–541. ISBN: 9781450371285. DOI: 10.1145/3377930.3389817.

[37] M. F. Sani. "Preprocessing event data in process mining." PhD thesis. RWTH Aachen University, Germany, 2023. URL: https://publications.rwth-aachen.de/record/963843.

[38] M. F. Sani, S. J. van Zelst, and W. M. P. van der Aalst. "Improving Process Discovery Results by Filtering Outliers Using Conditional Behavioural Probabilities." In: *Business Process Management Workshops - BPM 2017 International Workshops, Barcelona, Spain, September 10-11, 2017, Revised Papers*. Ed. by E. Teniente and M. Weidlich. Vol. 308. Lecture Notes in Business Information Processing. Springer, 2017, pp. 216–229. DOI: 10.1007/978-3-319-74030-0_16.

[39] M. F. Sani, S. J. van Zelst, and W. M. P. van der Aalst. "Repairing Outlier Behaviour in Event Logs using Contextual Behaviour." In: *Enterp. Model. Inf. Syst. Archit. Int. J. Concept. Model.* 14 (2019), 5:1–5:24. DOI: 10.18417/EMISA.14.5.

[40] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. "Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms." In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2013, pp. 847–855.

[41] E. Verbeek, J. C. A. M. Buijs, B. F. van Dongen, and W. M. P. van der Aalst. "ProM 6: The Process Mining Toolkit." In: *Proceedings of the Business Process Management 2010 Demonstration Track, Hoboken, NJ, USA, September 14-16, 2010*. Ed. by M. L. Rosa. Vol. 615. CEUR Workshop Proceedings. CEUR-WS.org, 2010. URL: https://ceur-ws.org/Vol-615/paper13.pdf.

[42] R. Vilalta, C. G. Giraud-Carrier, and P. Brazdil. "Meta-Learning - Concepts and Techniques." In: *Data Mining and Knowledge Discovery Handbook, 2nd ed.* Ed. by O. Maimon and L. Rokach. Springer, 2010, pp. 717–731. DOI: 10.1007/978-0-387-09823-4_36.

[43] J. D. Weerdt, M. D. Backer, J. Vanthienen, and B. Baesens. "A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs." In: *Inf. Syst.* 37.7 (2012), pp. 654–676. DOI: 10.1016/J.IS.2012.02.004.

[44] A. Weijters, W. van der Aalst, and A. de Medeiros. *Process mining with the Heuristics Miner algorithm.* Tech. rep. Technische Universiteit Eindhoven, 2006.

[45] A. Weijters and J. Ribeiro. "Flexible Heuristics Miner (FHM)." In: *2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM).* 2011, pp. 310–317. DOI: 10.1109/CIDM.2011.5949453.

[46] L. Wen, W. M. P. van der Aalst, J. Wang, and J. Sun. "Mining process models with non-free-choice constructs." In: *Data Min. Knowl. Discov.* 15.2 (2007), pp. 145–180. DOI: 10.1007/S10618-007-0065-Y.

[47] L. Wen, J. Wang, W. M. P. van der Aalst, B. Huang, and J. Sun. "Mining Process Models with Prime Invisible Tasks." In: *Data & Knowledge Engineering* 69.10 (2010), pp. 999–1021.

[48] S. J. van Zelst, B. F. van Dongen, and W. M. P. van der Aalst. "ILP-Based Process Discovery Using Hybrid Regions." In: *Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data, ATAED 2015, Satellite event of the conferences: 36th International Conference on Application and Theory of Petri Nets and Concurrency Petri Nets 2015 and 15th International Conference on Application of Concurrency to System Design ACSD 2015, Brussels, Belgium, June 22-23, 2015.* Ed. by W. M. P. van der Aalst, R. Bergenthum, and J. Carmona. Vol. 1371. CEUR Workshop Proceedings. CEUR-WS.org, 2015, pp. 47–61. URL: https://ceur-ws.org/Vol-1371/paper04.pdf.