Conditioning and Stability in Process Discovery

Anandi Karunaratne, Artem Polyvyanyy, and Alistair Moffat

The University of Melbourne, Australia
anandik@student.unimelb.edu.au, {artem.polyvyanyy; ammoffat}@unimelb.edu.au

Abstract. Process mining employs discovery algorithms to automatically derive process models from historical event logs. Despite decades of research in process discovery, practitioners still lack principled guidance on selecting algorithms for different purposes. Existing evaluation practices rely mainly on outcome-based measures, such as recall and precision, which assess model quality but overlook the characteristics of the discovery process itself – leaving an important question unanswered: *"Can I rely on the discovery algorithm?"* We propose a reliability analysis that characterizes process discovery algorithms through two properties inspired by numerical analysis: *algorithmic conditioning* and *algorithmic stability*. Algorithmic conditioning quantifies how changes in the input log propagate to the output model, assessing whether small perturbations lead to proportional model changes or erratic, unpredictable behavior. Algorithmic stability evaluates whether the algorithm remains reliable for a given analytical purpose, even when that purpose differs from its original design intent. Together, these properties enable principled assessment of algorithmic trustworthiness beyond outcome quality, addressing both inherent reliability and fitness for purpose. We formalize these non-functional properties and demonstrate their utility through an empirical study of popular discovery algorithms. Our evaluation reveals significant variation in how different perturbation types affect model properties, shows that algorithms can be stable for one analytical purpose while unstable for another, and quantifies the degree to which discovered models may deviate from ideal conditions. This shifts the focus from *how good* a discovered model is to *how reliable* the algorithm is for some specific analytical purposes; thereby providing a foundation for transparent, purpose-aware algorithm selection in process mining.

Keywords: process mining · process discovery · noise · stability · reliability

1 Introduction

Process mining bridges the gap between data and process analysis by extracting process-related insights from *event logs*. Within this field, *process discovery* focuses on automatically deriving a process model using the information recorded in an event log, enabling visualization, analysis, and improvement of operational processes [1].

Over the past two decades, numerous discovery algorithms have emerged, each optimizing for different goals: some favor simplicity, others behavioral accuracy or generalization beyond recorded traces [5]. Despite this variety, practitioners still lack principled guidance for selecting the right algorithm for their context [22, 24]. The central question is "*Can I trust this algorithm to reliably serve my analytical purpose?*"

Current evaluations focus on outcome-based measures such as recall, precision, generalization, and simplicity [1, 5]. While crucial, these measures assess the static *result* rather than the dynamic reliability of the *discovery process*. This outcome-centric view tells us *how good* a model is, but not the *reliability* of the algorithm. An algorithm that once performed well may behave unpredictably on a different log.

We argue that assessing algorithmic reliability requires examining two distinct facets of behavior. First, how do model properties respond to slight changes, or perturbations, in the input log? For instance, Inductive Miner aims to discover sound and fitting models [13] – but does it maintain these guarantees when the log contains noise? Beyond design goals, how do other properties change – does model size explode, do semantic patterns shift significantly? This perturbation-sensitivity analysis examines which model properties remain robust and which are compromised when faced with realistic log imperfections. While robustness to noise has been discussed in process mining literature, we reframe it systematically, asking "Which model properties are robust to which types of perturbations?" and quantifying the change. This nuanced view recognizes that an algorithm may preserve some properties while compromising others, and aligns robustness assessment with the specific properties relevant to the analyst's purpose. We term this *algorithmic conditioning*. Second, algorithms are often repurposed for analytical goals beyond their original design. A discovery algorithm optimized for producing simple, readable models might be repurposed by a compliance officer who needs to detect subtle infrequent process deviations. The algorithm may perform its designed task well, yet prove unsuitable for this alternative purpose. This raises questions about the algorithm's *stability* – its ability to serve a specific analytical purpose.

To formalize these two dimensions of reliability, we draw conceptual inspiration from numerical analysis, adapting – rather than directly mapping – two fundamental concepts: *conditioning* and *stability* [9, 11]. In that context, conditioning describes how sensitive a mathematical problem is to input perturbations, while stability describes how well an algorithm approximates the exact solution to a problem. We reinterpret these concepts to operate at the algorithmic level in process discovery, proposing two non-functional properties [10]:

- *Algorithmic conditioning*: How model properties of interest respond to log perturbations. Well-conditioned algorithms exhibit predictable, proportional changes in critical properties when the input log contains real-world imperfections such as noise or incompleteness.
- *Algorithmic stability*: Whether the algorithm remains reliable when applied to a specific analytical purpose. An algorithm may have high stability for one analytical goal but low stability for another.

These properties are distinct concerns: an algorithm may have excellent conditioning (predictable behavior under perturbations) but poor stability for a given analytical goal (unreliable when repurposed), or vice versa. By articulating them, we move beyond outcome-based evaluation toward a deeper understanding of algorithmic behavior. This perspective supports principled, purpose-aware algorithm selection in process mining.

Our contributions are twofold:

1. A conceptual framework formalizing two non-functional properties for process discovery algorithms: algorithmic conditioning (reliability under log perturbations) and algorithmic stability (reliability for analytical purposes).
2. An empirical demonstration of the applicability of the proposed framework using popular discovery algorithms, showing how these properties can guide algorithm selection in practice.

The remainder of this paper is structured as follows. Section 2 introduces the necessary preliminaries, allowing the framework to be formalized in Section 3, with Section 4 then presenting an empirical evaluation. Section 5 discusses related work and puts our proposals in to context, and Section 6 then concludes with future directions.

2 Preliminaries

This section lays the foundation for our exploration of reliability of process discovery algorithms.

2.1 Process Discovery

We begin by defining some process discovery concepts.

Systems. Let S be a system and A a finite set of activities it can perform. The set A^* represents all finite activity sequences (*traces*) over A. The system's language, $lang(S) \subseteq A^*$, is the set of traces S can generate, which may be infinite.

Event Logs. The operation of S yields a finite log $L \in \mathcal{L}$, where $\mathcal{L}$ is the space of logs and L is a multiset of traces. The language of L, $lang(L) \subset A^*$, is its support set $Supp(L)$. Ideally, it holds that, $lang(L) \subseteq lang(S)$. In practice, L may contain traces not in $lang(S)$.

Noise. Noise in event logs refers to behavior recorded in L that does not reflect the intended, controllable, and repeatable process execution. This includes infrequent events that are inaccurately recorded or caused by external factors. A noise function $n : \mathcal{L} \to \mathcal{L}$ perturbs the true process executions L_{true} to produce the actual log $L = n(L_{true})$. Types of noise include:

- Insertion: adding spurious events to a trace.
- Absence: removing legitimate events from a trace.
- Substitution: replacing events with incorrect ones.
- Ordering: changing event order of a trace.

Insertion and substitution noise can introduce activities not in A, leading to an effective alphabet expansion to $A' \supseteq A$, so that $lang(L) \subset A'^*$ [12].

Process Models. $M \in \mathcal{M}$ is the process model discovered from L, where $\mathcal{M}$ is the model space. The language of M, $lang(M) \subseteq A'^*$, may include traces in $lang(S)$, $lang(L)$, in both, or in neither. An event log is mapped to a certain process model by a process discovery algorithm. Examples of process models include Petri nets, BPMN models, and process trees.

2.2 Distance Measures

To quantify dissimilarity between models or between logs, we use distance functions on the model space, $d_{\mathcal{M}} : \mathcal{M} \times \mathcal{M} \to \mathbb{R}_{\geq 0}$, and on the log space, $d_{\mathcal{L}} : \mathcal{L} \times \mathcal{L} \to \mathbb{R}_{\geq 0}$. The measures below may serve as $d_{\mathcal{M}}$ or $d_{\mathcal{L}}$.

Structural Distance. Model complexity is measured by the total count of structural elements, with P and T denoting the sets of places and transitions in Petri net M [19]:

$$complexity(M) = |P| + |T| \,. \tag{1}$$

Structural dissimilarity between models M_1 and M_2 can be quantified as the normalized relative difference in complexity, measured with respect to M_2:

$$d_{struct}(M_1, M_2) = \frac{|complexity(M_1) - complexity(M_2)|}{complexity(M_2)} \,. \tag{2}$$

Language Jaccard Distance. The language Jaccard distance applies Jaccard distance [14] for behavioral comparison, providing a simple, set-based view of behavioral overlap. For any two artifacts X_1 and X_2 (either logs or models), it measures the dissimilarity of their (assumed finite) languages:

$$d_{lang}(X_1, X_2) = d_{Jaccard}(lang(X_1), lang(X_2)) \,, \tag{3}$$

where the Jaccard distance between two finite sets A and B is:

$$d_{Jaccard}(A, B) = 1 - {}^{|A \cap B|}\!/_{|A \cup B|} \,. \tag{4}$$

Precision and Recall. To capture directional relationships between two languages, we use precision and recall. For X_1 and X_2 (logs or models), recall (*rec*) is the fraction of X_1's behavior captured by X_2, and precision (*pre*) is the fraction of X_2's behavior supported by X_1 [17].

$$rec(X_1, X_2) = \frac{|lang(X_1) \cap lang(X_2)|}{|lang(X_1)|} \quad \text{and} \quad pre(X_1, X_2) = \frac{|lang(X_1) \cap lang(X_2)|}{|lang(X_2)|} \,. \tag{5}$$

These lead to two corresponding distance measures $d_{rec}(X_1, X_2) = 1 - rec(X_1, X_2)$ and $d_{pre}(X_1, X_2) = 1 - pre(X_1, X_2)$.

The F_1-*score* combines both measures:

$$F_1\text{-}score(X_1, X_2) = 2 \cdot \frac{pre(X_1, X_2) \cdot rec(X_1, X_2)}{pre(X_1, X_2) + rec(X_1, X_2)} \,. \tag{6}$$

The corresponding distance measure is $d_{F_1}(X_1, X_2) = 1 - F_1\text{-}score(X_1, X_2)$.

2.3 Conditioning and Stability in Numerical Analysis

We are now in a position to introduce the numerical analysis concepts that are used in our framework [9, 11].

Conditioning. Conditioning measures how sensitive a problem's solution is to small changes in the input. A problem is *well-conditioned* if small input perturbations cause proportionally small changes in the exact solution, and *ill-conditioned* otherwise. For a scalar function $y = f(x)$, and for a small perturbation Δx, producing a corresponding change Δy in the output, we identify κ such that:

$$\frac{|\Delta y|/|y|}{|\Delta x|/|x|} \leq \kappa\,, \text{ where } \kappa \in \mathbb{R}_{\geq 0}\,. \tag{7}$$

The condition coefficient κ quantifies how much relative input errors can be amplified in the output: values near 1 indicate a well-conditioned problem, while large values indicate an ill-conditioned problem.

Stability. Stability assesses a numerical algorithm's resilience to small errors (e.g., rounding or measurement inaccuracies) in input or execution, ensuring reliable results. It is categorized into forward stability and backward stability. *Forward stability* examines whether the computed output is close to the exact solution. *Backward stability* examines whether the computed output equals the exact solution to a slightly perturbed problem. We focus on forward stability because backward stability is ill-suited to process discovery: the algorithm that maps event logs to process models involves an inherent many-to-many relationship, meaning the "exact solution" for a given log is non-unique and the input log space is discrete, violating the necessary continuous input and unique solution assumptions of numerical stability analysis.

Forward stability assesses how sensitive an algorithm's output is to its execution. Formally, let f denote the exact function mapping input x to its true output $f(x)$, and let $\hat{f}$ denote the algorithmic approximation that produces $\hat{f}(x)$. An algorithm is *forward stable* if for all relevant inputs, forward error, $\Delta y = \hat{f}(x) - f(x)$, satisfies:

$$|\Delta y| \leq \epsilon\,, \text{ so that } \hat{f}(x) = f(x) + \Delta y\,, \tag{8}$$

where ϵ is a small positive threshold bounding the output error.

3 Reliability of Discovery Algorithms

Building on the concepts of Section 2.3, this section formalizes conditioning and stability of process discovery algorithms. A key requirement is that process discovery algorithms must contend with two fundamental challenges: real-world event logs contain noise and imperfections, and analysts apply algorithms to diverse purposes beyond their original design intent.

Figure 1 illustrates this setting. The core task involves a discovery algorithm $\hat{f} : \mathcal{L} \to \mathcal{M}$ that maps an event log $L \in \mathcal{L}$ to a process model $\hat{f}(L)$. This relationship is inherently challenged by two fundamental realities. First, real-world logs are rarely ideal and are typically affected by noise, which we represent as a perturbation function $n : \mathcal{L} \to \mathcal{L}$. Second, there is a risk of analytical goal misalignment: the algorithm's output $\hat{f}(L)$ may not suit the specific purpose required by the analyst.

To formally measure this reliability gap, we introduce the *ideal* function $f : \mathcal{L} \to \mathcal{M}$, which conceptually captures the model that perfectly aligns with the analyst's objective. To quantify these challenges, we assume the existence of distance functions $d_{\mathcal{L}} : \mathcal{L} \times \mathcal{L} \to \mathbb{R}_{\geq 0}$ and $d_{\mathcal{M}} : \mathcal{M} \times \mathcal{M} \to \mathbb{R}_{\geq 0}$, which reflect the dissimilarity between logs and models, respectively. The specific choice of these metrics determines precisely which aspects of the algorithm's behavior get measured. These two axes – sensitivity to perturbations and alignment with intent – inform the overall reliability of a discovery algorithm. We formalize these concerns through two properties: algorithmic conditioning and algorithmic stability.

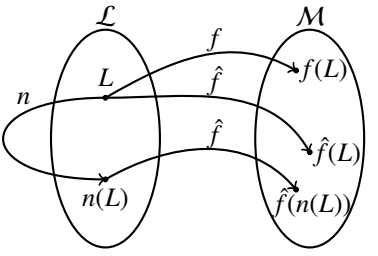


Fig. 1: Process discovery. Algorithm $\hat{f}$ maps log L and perturbed log $n(L)$ to models $\hat{f}(L)$ and $\hat{f}(n(L))$. Log changes should produce predictable model changes. Ideal algorithm f aligns with analyst's purpose; we seek $f(L)$ but obtain $\hat{f}(L)$.

3.1 Algorithmic Conditioning

Algorithmic conditioning quantifies how changes in the input log propagate to changes in the output model. It addresses the first challenge of process discovery, that is, response to imperfect data.

Definition 1 (Condition Number). *Given distance measures $d_{\mathcal{L}}$ and $d_{\mathcal{M}}$, the* condition number *of discovery algorithm $\hat{f}$ for log L is:*

$$\kappa_{\hat{f}}(L) = \sup_{\substack{n:\mathcal{L}\to\mathcal{L} \\ d_{\mathcal{L}}(L,n(L))\leq\delta}} \frac{d_{\mathcal{M}}(\hat{f}(L), \hat{f}(n(L)))}{d_{\mathcal{L}}(L, n(L))}\,, \tag{9}$$

where n is a perturbation function, $\delta > 0$ bounds the perturbation size, and the supremum is taken over all perturbations within this bound.

The conditioning of algorithm $\hat{f}$ *over input space $\mathcal{L}$ is:*

$$\kappa_{\hat{f}} = \sup_{L\in\mathcal{L}} \kappa_{\hat{f}}(L)\,. \tag{10}$$

An algorithm is perfectly conditioned *if $\kappa_{\hat{f}} = 0$, indicating that the discovered model remains unchanged under perturbations. An algorithm is* well-conditioned *if $\kappa_{\hat{f}}$ is small (close to 1), indicating that relative input changes produce proportional output changes. An algorithm is* ill-conditioned *if $\kappa_{\hat{f}}$ is substantially greater than 1, indicating that the algorithm amplifies small input perturbations into disproportionate output changes.* ⌟

The condition number $\kappa_{\hat{f}}$ depends critically on the choice of distance measures $d_{\mathcal{L}}$ and $d_{\mathcal{M}}$. Different choices capture different aspects of algorithmic behavior. For instance, $d_{\mathcal{M}}$ based on structural differences (e.g., edit distance between process models) captures whether the algorithm produces structurally similar models under perturbations, while

$d_{\mathcal{M}}$ based on behavioral properties (e.g., differences in fitness or precision) captures whether the algorithm maintains consistent quality of discovered models.

An algorithm may exhibit good conditioning with respect to one pair of distance measures while showing poor conditioning with respect to another. For example, Inductive Miner might maintain consistent fitness scores under log perturbations (well-conditioned for fitness-based $d_{\mathcal{M}}$) but produce models of varying structural complexity (ill-conditioned for structure-based $d_{\mathcal{M}}$). This multifaceted nature of conditioning reflects the fact that algorithms preserve different properties at different levels of reliability.

The condition number provides a quantitative lens for assessing whether an algorithm can be trusted to behave predictably when faced with real-world data imperfections. A well-conditioned algorithm offers practitioners confidence that the discovered model reflects stable process characteristics rather than artifacts of data quality issues.

3.2 Algorithmic Stability

Algorithmic stability assesses whether an algorithm reliably serves a specific analytical purpose. It addresses the second challenge of process discovery, that is, alignment with analytical goals. This property recognizes that practitioners often apply discovery algorithms to tasks beyond what they were originally designed for, and evaluates whether the algorithm consistently produces suitable results.

The key insight is to compare the actual algorithm $\hat{f}$ against an ideal function $f : \mathcal{L} \to \mathcal{M}$ that represents the analyst's goal. This ideal function encodes what the "perfect" algorithm would produce for the specific purpose. Then, stability quantifies how closely the actual algorithm $\hat{f}$ approximates this ideal across different input logs.

Definition 2 (Stability). *Given $\epsilon \geq 0$ and distance measure $d_{\mathcal{M}}$, algorithm $\hat{f}$ is* ϵ-stable *with respect to ideal function f if for all logs $L \in \mathcal{L}$, it holds that:*

$$d_{\mathcal{M}}(f(L), \hat{f}(L)) \leq \epsilon\,. \tag{11}$$

⌟

A small ϵ indicates the algorithm consistently produces models close to ideal, making it reliable for the intended analysis. A large ϵ signals that the algorithm's outputs deviate significantly from what we need, rendering it unsuitable regardless of data quality.

Stability is inherently purpose-dependent: the same algorithm may exhibit high stability for one analytical goal (small ϵ with respect to one ideal function f_1) while showing low stability for another goal (large ϵ with respect to a different ideal function f_2). For instance, an algorithm optimized for discovering simple, abstract process overviews may be highly stable for conformance checking purposes but exhibit poor stability when the goal is to detect rare process variants or subtle compliance violations.

The choice of ideal function f and distance measure $d_{\mathcal{M}}$ must align with the specific analytical purpose. If the goal is to discover models with high precision, then f might represent an algorithm that maximizes precision, and $d_{\mathcal{M}}$ would measure precision differences. If the goal is to find simple models, then f represents an algorithm producing minimally complex models, and $d_{\mathcal{M}}$ would capture complexity differences.

These dimensions are independent: an algorithm can exhibit good conditioning but poor stability for a particular purpose, or vice versa. Together, they provide a complete picture of algorithmic reliability. Good conditioning without stability means predictable but unsuitable results; good stability without conditioning means appropriate results that are unreliable for imperfect data. Only when both properties hold can an algorithm be trusted to reliably serve its analytical purpose when applied to real-world data.

4 Evaluation

This empirical study focuses on demonstrating algorithmic reliability assessment, through conditioning and alignment assessment with analytical goals[1].

4.1 Design

We selected three candidate algorithms for evaluation, namely the standard variants of Alpha Miner [2], Heuristics Miner [23], and Inductive Miner [13], with that choice based on the four criteria: implementation in PM4Py[4] for reproducibility and standardization; production of Petri net models or comparable formal representations; wide adoption; and representative of diverse algorithmic families [1]. We aim to evaluate the conditioning of structural deviation of resulting models under log perturbations for those algorithms.

The analytical objectives to check the alignment for this evaluations are:

1. Rediscoverability: Discover a model accurately representing the system S that generated the observed log L; and
2. Behavior replay: Produce a model in which log behavior is fully replayable, ensuring fitness with observed executions.

We employed the token-based replay approach [18] as implemented in PM4Py to compute d_{rec} and d_{pre}; these primary measures were subsequently used to calculate the corresponding d_{F_1} (see Eq. (6)).

4.2 Dataset Generation

We employed three publicly available event logs as base systems: Sepsis (1,050 traces) [15], RTFMS (150,370 traces) [7], and BPIC2012 (13,087 traces) [21]. We treat these original logs as ground-truth representations of their respective systems.

From each base log, we generated samples of seven sizes (1,000, 2,000, 4,000, 10,000, 20,000, 40,000, and 100,000 traces) by sampling with replacement. This produced 21 clean logs, capturing different behavioral profiles of the underlying systems.

Noise was systematically introduced using the tool *SNIP* [12], which implements four literature-informed noise types: absence, insertion, ordering, and substitution. Each noise type can manifest through multiple strategies – for instance, ordering noise may result from swapping adjacent activities or shifting an activity to a different position.

[1] Experimental resources are available at github.com/AnandiKarunaratne/stability_codebase.

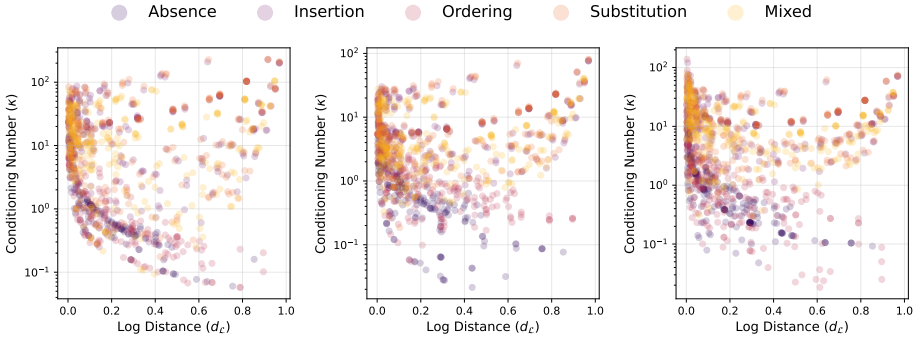
SNIP randomly selects among these strategies when applying each perturbation, ensuring that a single noise type captures the full range of ways that distortion can occur in practice, based on scenarios documented in process mining literature. We also included a mixed type combining all four mechanisms. Noise was injected through sampling with replacement: for i iterations, a trace is selected, perturbed with one operation, and reinserted. This reflects realistic noise accumulation in event logs.

We applied noise at seven intensity levels: 0.1%, 0.2%, 0.4%, 1.0%, 2.0%, 4.0%, and 10.0%, where the percentage indicates i as a proportion of log size (e.g., 1.0% of 1,000 traces = 10 injections). This yielded 35 noisy variants per clean log. To ensure robustness, noise generation was repeated five times per configuration, producing 3,696 logs: 21 clean and 3,675 noisy variants.

Each algorithm was applied to logs in the dataset. Due to computational constraints, we adopted stratified random sampling ensuring: (1) complete coverage of all 21 clean logs, which serve as benchmarks for stability assessment; and (2) representative sampling of noisy logs across noise types, levels, and log sizes. We have discovered 1,253 models per algorithm, totaling 3,759 models with 33.52% noisy log coverage. This dataset is available at https://doi.org/10.26188/30739082.

4.3 Conditioning Analysis

We empirically assess algorithmic conditioning by evaluating how structural changes (measured by d_{struct}) in discovered models respond to behavioral dissimilarity between the logs (measured by d_{lang}). We consider each clean and noisy log pair generated, which introduced a specific controlled log perturbations, measure the d_{struct} and d_{lang} values for them, and calculate the local κ for each instance.



(a) Alpha Miner (b) Heuristics Miner (c) Inductive Miner

Fig. 2: Local condition numbers (κ) across log perturbations for each process discovery algorithm. Each point represents a clean-noisy log pair, with κ quantifying the structural change in the discovered models, captured by d_{struct}, relative to the difference between logs (measured by d_{lang}). Point color indicates perturbation type.

Table 1: Conditioning statistics by algorithm and noise type. Heuristics Miner (underlined) consistently shows the lowest $\kappa_{\hat{f}}$ values. *Indicates validation failure (relative error ≥ 50%); these values are excluded from the main analysis.

Noise type	Alpha			Heuristics			Inductive		
	$\kappa_{\hat{f}}$	κ_{50}	κ_{95}	$\kappa_{\hat{f}}$	κ_{50}	κ_{95}	$\kappa_{\hat{f}}$	κ_{50}	κ_{95}
Absence	11.23	0.28	2.98	<u>7.36</u>	0.36	2.39	14.16	0.65	6.65
Insertion	225.96	17.54	82.82	<u>76.66</u>	5.58	26.07	120.31	11.12	48.81
Ordering	27.69*	0.34*	6.53*	<u>21.27</u>	0.78	9.04	71.29*	0.74*	14.70*
Substitution	230.47	18.94	103.02	<u>79.81</u>	5.39	35.42	72.51	12.28	43.62
Mixed	104.47	11.69	54.27	<u>40.39</u>	3.42	19.31	74.13	9.32	41.84

The relationship between κ values and d_{lang} (Fig. 2) reveals that while hypersensitivity to small log changes is consistently observed across all algorithms, with κ values concentrated at higher levels when $d_{lang} \approx 0$, subsequent behavior exhibits distinct patters across perturbation types.

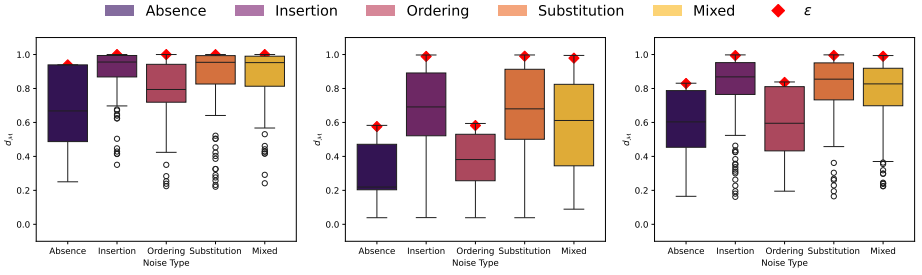
Absence and ordering perturbations show clear decay: maximum κ declines from over 10 at low d_{lang} to near zero at high d_{lang}. These perturbations directly disrupt control-flow dependencies, driving models toward degenerate structures (e.g., flower-like Petri nets). Once structural collapse occurs, further perturbations yield minimal divergence. In contrast, insertion, substitution, and mixed perturbations exhibit non-monotonic behavior: maximum κ values rebound at large d_{lang} (above 0.8), reaching extreme sensitivity comparable to low d_{lang} levels. This occurs because these perturbations introduce new activities, forcing models to add transitions and maintain or increase structural complexity rather than collapse.

Given this observation, we computed three statistics: (1) $\kappa_{\hat{f}} = \max \kappa$ capturing worst-case sensitivity; (2) κ_{50} (median) representing typical behavior; and (3) κ_{95} (95th percentile) indicating extreme but non-outlier sensitivity. The ratio κ_{95}/κ_{50} quantifies input-dependency: values exceeding 10 indicate conditioning varies by more than an order of magnitude. We validated estimates using a 70–30 split; errors below 50% confirm generalization. We analyze only validated results (that is, excluding Alpha Miner and Inductive Miner under ordering noise). Results appear in Table 1.

Three trends emerge. First, all algorithms are severely ill-conditioned ($\kappa_{\hat{f}}$ ranging from 7.36 to 230.47), with the highest sensitivity for Alpha Miner under substitution noise. Second, input-dependency concentrates in specific perturbations: ordering and absence show $\kappa_{95}/\kappa_{50} > 10$, where $\approx 5\%$ of logs experience catastrophic sensitivity, while insertion and substitution maintain stable ratios, potentially because new activities consistently increase model complexity. Finally, Heuristics Miner demonstrates superior conditioning (κ_{50} down to 0.36, $\kappa_{\hat{f}} \leq 79.81$) compared to Alpha Miner ($\kappa_{\hat{f}}$ up to 230.47) and Inductive Miner ($\kappa_{\hat{f}}$ up to 120.31), suggesting its frequency-based filtering offers measurable noise tolerance.

4.4 Stability Analysis

We now quantify algorithmic stability with respect to each analytical objective.



(a) Alpha Miner (b) Heuristics Miner (c) Inductive Miner

Fig. 3: Distribution of model distance $d_{\mathcal{M}}$ from the ground truth system S by algorithm and noise type for rediscoverability goal. Red points indicate 95% prediction bounds ϵ.

Goal 1: Rediscoverability. We seek to discover the underlying system S from observed log L such that $lang(f(L)) = lang(S)$. We define $d_{\mathcal{M}}$ as the distance between discovered model and ground truth: $d_{\mathcal{M}}(\hat{f}(L), S) = d_{F_1}(lang(\hat{f}(L)), lang(S))$. Input distance $d_{\mathcal{L}}$ captures log-level perturbations via d_{lang}.

To empirically assess stability, we compare the ideal model $f(L) = S$ (ground truth) against the discovered model $\hat{f}(L)$ for each log in our dataset. We compute $d_{\mathcal{M}}(S, \hat{f}(L))$ using d_{F_1} between their languages.

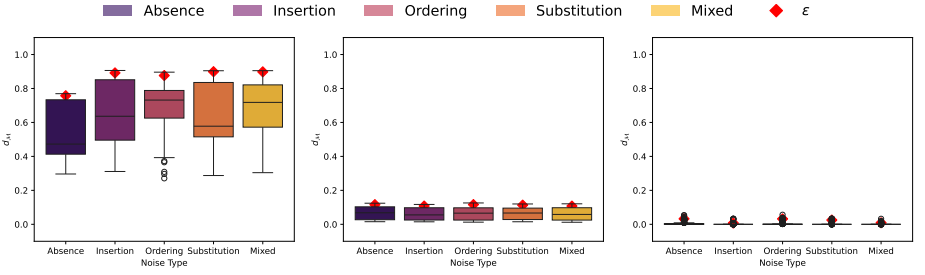
To generalize our findings to unseen logs, we use bootstrap prediction intervals. From our sample of l logs per noise type, we resample with replacement $B = 10{,}000$ times, computing the 95th percentile of $d_{\mathcal{M}}$ values for each bootstrap sample. The 95th percentile of these bootstrap percentiles gives us an upper prediction bound ϵ.

This allows us to state that with 95% confidence, for any new log L from the same distribution containing the same perturbation type, there is a 95% probability that $d_{\mathcal{M}}(f(L), \hat{f}(L)) < \epsilon$. Figure 3 shows the box plot showing the distribution of $d_{\mathcal{M}}$ values and our calculated ϵ values.

All algorithms exhibit severe instability ($\epsilon > 0.8$) across most noise types, with insertion noise causing near-complete or complete model collapse ($\epsilon \approx 1.0$) universally across Inductive ($\epsilon = 0.994$), Alpha ($\epsilon = 1.0$), and Heuristics ($\epsilon = 0.989$). Substitution noise similarly destabilizes all algorithms ($\epsilon > 0.99$), while ordering noise shows variable impact, with Heuristics demonstrating relative robustness ($\epsilon = 0.582$) compared to Inductive ($\epsilon = 0.836$) and Alpha ($\epsilon = 1.0$). Only absence noise reveals differential algorithm behavior: Heuristics achieves the tightest bound ($\epsilon = 0.576$), followed by Inductive ($\epsilon = 0.830$), while Alpha remains highly unstable ($\epsilon = 0.939$). Mixed noise, combining multiple perturbation types, consistently produces high instability ($\epsilon > 0.97$) across all algorithms. These discovery algorithms are fundamentally unstable under common log perturbations for our intended goal.

Goal 2: Behavior Replay. We now consider the ideal algorithm f as one that identifies a model perfectly replaying the input log: $d_{rec}(L, f(L)) = 0$.

Inductive Miner guarantees by design that $d_{rec}(L, \hat{f}(L)) = 0$ for any log L. This yields, $d_{\mathcal{M}} = |d_{rec}(L, \hat{f}(L)) - d_{rec}(n(L), \hat{f}(n(L)))| = |0 - 0| = 0$, making Inductive Miner



(a) Alpha Miner (b) Heuristics Miner (c) Inductive Miner

Fig. 4: Distribution of model distance $d_{\mathcal{M}}$ from perfect replay fitness by algorithm and noise type for behavior replay goal. Red point indicates 95% prediction bound ϵ.

perfectly conditioned with a condition number $\kappa_{\hat{f}} = 0$ (data perturbations cause no model deviations with respect to replay fitness). Hence, Inductive should exhibit stability for our objective. We verify this empirically and test whether other algorithms can reliably achieve the same replay guarantee.

For stability, we measure $d_{\mathcal{M}} = d_{rec}(L, \hat{f}(L))$: the deviation from perfect replay fitness. We apply bootstrap prediction intervals (as in Goal 1) to establish upper bounds ϵ such that, with 95% confidence, 95% of future logs will have $d_{\mathcal{M}} < \epsilon$.

Figure 4 shows stability bounds for the replay fitness objective. Inductive Miner demonstrates near-perfect stability ($\epsilon \approx 0.007$–0.032), validating its design guarantee that $d_{rec}(L, \hat{f}(L)) = 0$ regardless of perturbations. The slight non-zero values likely reflect approximation errors in the utilized token-based d_{rec} implementation rather than algorithmic instability. Insertion and mixed noise show the tightest bounds ($\epsilon = 0.007$ and 0.009 respectively), while ordering and absence noise exhibit slightly higher but still negligible instability ($\epsilon \approx 0.032$).

Heuristics Miner shows strong stability ($\epsilon \approx 0.107$–0.116) with remarkable consistency across all perturbation types, indicating modest replay fitness degradation. In contrast, Alpha Miner exhibits severe instability across all noise types, with absence noise showing the lowest ($\epsilon = 0.757$) and substitution the highest ($\epsilon = 0.899$) instability, making it unsuitable for replay-focused applications.

5 Related Work

Our proposed properties builds upon and extends several established research directions in process mining, particularly work on log quality, noise resilience, and rediscoverability guarantees.

The concept of algorithmic conditioning relates closely to existing work on log quality and noise resilience in process mining. The Heuristics Miner [23] was explicitly designed to handle noisy logs by computing dependency measures between activities based on observed frequencies. It introduces three configurable thresholds (dependency threshold, positive observations threshold, and relative-to-best threshold) to filter low-

confidence dependencies while preserving frequent behavioral patterns. By calculating reliability scores for causal relations based on frequency patterns, the algorithm can correctly mine causal structures even in the presence of noise. Vanden Broucke and De Weerdt [20] refined this approach with improved robustness to noise, the ability to discover duplicate activities, and flexible configuration options to drive discovery according to end-user preferences, addressing several identified limitations in earlier heuristic miners.

Complementary approaches focus on preprocessing event logs before discovery. Conforti et al. [6] developed an automated technique to filter infrequent behavior by constructing a log automaton and identifying anomalies through alignment-based replay. Their evaluation showed that even low levels of infrequent behavior (as little as 2% of the total log) can have detrimental effects on model quality for algorithms like Heuristics Miner, Fodina [20], and Inductive Miner. Fani Sani et al. [8] proposed filtering outliers using conditional behavioral probabilities, exploiting the observation that infrequent behavior often exhibits different conditional dependencies than normal process execution. Măruşter et al. [16] applied supervised learning techniques to enhance the Alpha algorithm's noise tolerance by learning rules that distinguish genuine process behavior from noise.

These works provide valuable practical solutions for handling noisy logs. Our conditioning framework complements existing noise robustness work by systematically characterizing perturbation sensitivity across model properties and perturbation types. Where noise robustness research establishes mechanisms (filtering, thresholds, preprocessing) for handling imperfect logs, conditioning quantifies the resulting sensitivity landscape: which properties are affected by which perturbations, and to what degree.

The concept most closely related to our stability notion is *rediscoverability*: whether a discovery algorithm can reconstruct the generative system using a log.

Van der Aalst et al. [2] introduced the Alpha algorithm with formal guarantees for isomorphic-rediscoverability: the discovered Petri net is structurally identical to the original system under specific assumptions. The key requirement is a directly-follows complete log containing exhaustive information about local activity orderings. However, this assumption is often unrealistic, and the algorithm cannot handle short loops or non-free-choice structures, limiting practical applicability.

Leemans [13] formalized language-rediscoverability for the Inductive Miner framework, which discovers block-structured models (process trees) through recursive log splitting. The framework guarantees that if a process is expressible as a Process Tree and the log is directly-follows complete (revealing behavioral dependencies for cut-based decomposition), the algorithm rediscovers a model accepting the same language as the original system. It is demonstrated that these guarantees hold efficiently even for large logs.

Recent work extends rediscoverability to object-centric settings with multiple interacting object types. Barenholz et al. [3] introduce reconstructability for Typed Jackson Nets (T-JNs), proving that if traditional – that is, neither object-centric nor agent-based – miners rediscover individual projected models from object-specific views, the T-JN composition operator guarantees bisimilar rediscoverability of the integrated system.

Stability broadens rediscoverability's perspective from theoretical system reconstruction to practical fitness for analytical purpose. While rediscoverability asks "Can we reconstruct the original system given complete logs?", stability asks "Can the algorithm reliably serve my specific analytical purpose given realistic log conditions?" This shift recognizes that practitioners often use discovery for diverse purposes beyond system reconstruction-conformance checking.

6 Conclusion

This work introduces two non-functional properties for evaluating the reliability of process discovery algorithms through conditioning and stability analysis, drawing from numerical analysis to characterize algorithmic behavior under input perturbations and purpose alignment.

Conditioning quantifies how sensitive an algorithm is to input log perturbations through the condition number $\kappa_{\hat{f}}$, revealing how much a model property changes under minor log perturbations, helping to understand how much should we care about data preprocessing. Stability analysis addresses algorithm repurposing, quantifying expected deviations when an algorithm designed for objective is evaluated against a different analytical goal. Our stability property, which measures the maximum deviation from the analytical goal, determines usability, whether results are adequate for analysis.

Our empirical evaluation demonstrated the usability of these properties using three widely-used algorithms (Alpha, Heuristics, and Inductive miners), conditioning for structural deviations of models under perturbations and stability across two objectives provides quantifiable risk factors for expected model deviation from the expected. The analysis reveals that stability is fundamentally objective-dependent: an algorithm cannot be deemed "stable" or "reliable" in isolation but must be assessed relative to specific analytical goals. The same algorithm may exhibit perfect stability for one objective while showing severe instability for another, depending on the alignment between its design objective and the evaluation goal.

This objective-dependence has critical practical implications. Practitioners must carefully match algorithm selection to their intended analytical purpose, recognizing that algorithms optimized for one objective may be fundamentally unsuitable for another. Research findings based on misaligned algorithm selections can yield misleading conclusions about algorithmic quality and performance. Importantly, once stability analysis confirms that algorithm repurposing is acceptable for a given objective (that is, stability bounds are within tolerable ranges), practitioners can then confidently consider other non-functional properties such as computational efficiency, scalability, and memory usage to balance accuracy against performance requirements. Without establishing stability first, optimizing for speed or resource efficiency may be premature – fast algorithms that produce unreliable results may offer limited practical value.

Future research will primarily focus on expanding and operationalizing these properties. This involves extending its scope by applying it to a wider array of process discovery algorithms (like genetic or deep learning miners) to create a comprehensive reliability benchmark for the field. As process mining continues its evolution toward widespread industrial deployment, understanding algorithmic reliability through

objective-specific stability analysis becomes essential for trustworthy automated process discovery in practice. Within the broader context of Information Systems Engineering, this work contributes a principled foundation for assessing the reliability of automated process modeling techniques that inform system design, analysis, and evolution.

References

[1] van der Aalst, W.M.P.: Process Mining — Data Science in Action. Springer, 2 edn. (2016)
[2] van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Trans. Knowl. Data Eng. **16**, 1128–1142 (2004)
[3] Barenholz, D., Montali, M., Polyvyanyy, A., Reijers, H.A., Rivkin, A., van der Werf, J.M.E.M.: There and back again: On the reconstructability and rediscoverability of typed Jackson nets. In: Appl. Theory Petri Nets Concurr., pp. 37–58, Springer (2023)
[4] Berti, A., van Zelst, S., Schuster, D.: PM4Py: A process mining library for Python. Software Impacts **17**, 100556:1–100556:7 (2023)
[5] Buijs, J.C.A.M., van Dongen, B., van der Aalst, W.M.P.: Quality dimensions in process discovery: The importance of fitness, precision, generalization and simplicity. Int. J. Coop. Inf. Syst. **23**, 1440001:1–1440001:39 (2014)
[6] Conforti, R., La Rosa, M., ter Hofstede, A.H.M.: Filtering out infrequent behavior from business process event logs. IEEE Trans. Knowl. Data Eng. **29**, 300–314 (2017)
[7] De Leoni, M., Mannhardt, F.: Road traffic fine management process (2015), doi: 10.4121/UUID:270FD440-1057-4FB9-89A9-B699B47990F5
[8] Fani Sani, M., van Zelst, S.J., van der Aalst, W.M.P.: Improving process discovery results by filtering outliers using conditional behavioural probabilities. In: Lecture Notes in Business Information Processing, pp. 216–229, Springer (2018)
[9] Ford, W.: Conditioning of problems and stability of algorithms. In: Numerical Linear Algebra with Applications, pp. 181–204, Elsevier (2015)
[10] Hamlet, D.: Functional vs. non-functional properties. In: Composing Software Components, pp. 311–318, Springer (2010)
[11] de Jong, L.S.: Towards a formal definition of numerical stability. Numerische Mathematik **28**, 211–219 (1977)
[12] Karunaratne, A., Polyvyanyy, A., Moffat, A.: The effects of log noise in process mining. IEEE Access **13**, 198540–198563 (2025)
[13] Leemans, S.J.J.: Robust Process Mining with Guarantees: Process Discovery, Conformance Checking and Enhancement. Springer (2022)
[14] Levandowsky, M., Winter, D.: Distance between sets. Nature **234**, 34–35 (1971)
[15] Mannhardt, F.: Sepsis cases – event log (2016), doi: 10.4121/UUID:915D2BFB-7E84-49AD-A286-DC35F063A460
[16] Măruşter, L., Weijters, A.J.M.M., van der Aalst, W.M.P., van den Bosch, A.: A rule-based approach for process discovery: Dealing with noise and imbalance in process logs. Data Min. Knowl. Discov. **13**, 67–87 (2006)
[17] Polyvyanyy, A., Solti, A., Weidlich, M., Di Ciccio, C., Mendling, J.: Monotone precision and recall measures for comparing executions and specifications of dynamic systems. ACM Trans. Softw. Eng. Methodol. **29**, 17:1–17:41 (2020)
[18] Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. Inf. Syst. **33**, 64–95 (2008)
[19] Schalk, P., Burke, A., Lorenz, R.: Navigating complexity: Comparing complexity measures with Weyuker's properties. In: ICPM, pp. 145–152, IEEE (2024)

[20] vanden Broucke, S.K.L.M., De Weerdt, J.: Fodina: A robust and flexible heuristic process discovery technique. Decis. Support Syst. **100**, 109–118 (2017)
[21] van Dongen, B.: BPI challenge 2012 (2012), doi: 10.4121/UUID:3926DB30-F712-4394-AEBC-75976070E91F
[22] Wang, J., Wong, R.K., Ding, J., Guo, Q., Wen, L.: Efficient selection of process mining algorithms. IEEE Trans. Serv. Comput. **6**, 484–496 (2013)
[23] Weijters, A.J.M.M., van der Aalst, W.M.P., Alves De Medeiros, A.K.: Process mining with the HeuristicsMiner algorithm. BETA publicatie : working papers, TU/e (2006)
[24] van der Werf, J.M.E.M., Polyvyanyy, A., van Wensveen, B.R., Brinkhuis, M., Reijers, H.A.: All that glitters is not gold: Four maturity stages of process discovery algorithms. Inf. Syst. **114**, 102155:1–102155:12 (2023)