DAFDiscover: Robust Mining Algorithm for Dynamic Approximate Functional Dependencies on Dirty Data

Xiaoou Ding
dingxiaoou@hit.edu.cn
Harbin Institute of Technology

Yixing Lu
2021110839@stu.hit.edu.cn
Harbin Institute of Technology

Hongzhi Wang
wangzh@hit.edu.cn
Harbin Institute of Technology

Chen Wang
wang_chen@tsinghua.edu.cn
Tsinghua University, China

Yida Liu
22B903040@stu.hit.edu.cn
Harbin Institute of Technology

Jianmin Wang
jimwang@tsinghua.edu.cn
Tsinghua University, China

ABSTRACT

Data dependency mining plays a crucial role in understanding data relationships. To address the increasing complexities of real-world data, Approximate Functional Dependencies (AFDs) have been introduced, building upon traditional FD. However, existing AFD approaches use static relaxation coefficients, limiting their effectiveness in capturing dependencies in noisy data. We propose a dynamic AFD variant, DAFD, which incorporates attribute error rates. We establish a bijection between DAFD and FD, develop its inference system, and introduce DAFDiscover, an algorithm for mining dependencies directly on noisy data. DAFDiscover matches the time and space complexity of SOTA AFD mining methods while offering superior performance. We theoretically prove its correctness, provide a method for calculating DAFD probabilities (DAFD-*prob*), and derive a lower bound for DAFD's validity on dirty data. Experimental results on multiple public datasets demonstrate the semantic superiority of DAFD and the effectiveness of DAFDiscover compared to existing SOTA AFD mining techniques.

PVLDB Reference Format:
Xiaoou Ding, Yixing Lu, Hongzhi Wang, Chen Wang, Yida Liu, and Jianmin Wang. DAFDiscover: Robust Mining Algorithm for Dynamic Approximate Functional Dependencies on Dirty Data. PVLDB, XX(X): XXX-XXX, 2024.

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/agithuber2023/DAFDiscover.

1 INTRODUCTION

Ensuring a high degree of reliability in data quality rules is essential for achieving effective data cleaning results. Automatic discovery techniques have been developed to fully uncover the implicit dependencies from large-scale historical data, which also extract valuable hidden rule models from the data [8, 16, 30]. Functional dependencies (FDs) [12, 15], which constitute a pivotal category of integrity constraint languages, have witnessed the implementation of diverse and efficient mining methodologies. These have been employed in tasks encompassing violation detection, data repair, and schema normalization, thereby facilitating a more comprehensive and rigorous approach to data quality management (see surveys [7, 30]).

Conventionally, exact FDs are utilized to describe the relationship models between data. To enhance the applicability of integrity constraints in big data, exact FDs have been relaxed from two perspectives [7]: extent of *satisfaction* and *attribute comparison*. Specifically, a relaxed functional dependency (RFD) can be concisely represented as $X_{\Phi_1} \xrightarrow{\Psi \leq \epsilon} Y_{\Phi_2}$. Here, X and Y represent attribute sets, while Φ_1 and Φ_2 capture the relaxation in terms of comparison methods. Ψ reflects the relaxation approach in terms of satisfaction degree, and ϵ denotes the relaxation limit in satisfaction. The relaxation Φ_1, Φ_2 in comparison methods enables RFDs to express dependency relationships in heterogeneous data, while the relaxation in satisfaction degree (i.e., $\Psi \leq \epsilon$) allows RFDs to express dependency relationships in low-quality. Such dependencies can be referred to as approximate functional dependencies (AFD). data. Compared to exact FDs, RFDs demonstrate superior adaptability in addressing the growing demands in real-world scenarios. meeting the growing demands of real-world scenarios.

For the mechanism provided by AFD:$X \xrightarrow{\Psi \leq \epsilon} Y$, which utilizes a fixed hyperparameter ϵ to enhance tolerance of noise in large datasets, there are still limitations to this static characterization of tolerance in the face of widespread dirty data in real-world scenarios. Firstly, dirty data is often unevenly distributed across datasets, and a uniform tolerance level cannot adapt to varying proportions of dirty data that change with dependency patterns. This results in difficulty achieving accurate performance for mining results. Secondly, the uniform tolerance level fails to adapt to changes in the number of dependent attributes. Even if the proportion of dirty data is the same across all attributes, a fixed ϵ cannot accommodate dependency candidates with different numbers of attributes. This leads mining algorithms to tend towards either missing multi-attribute dependencies or misjudging dependencies with fewer attributes. We illustrate this issue with a motivated example below.

Example 1.1. Table 1 presents monitoring records for an electrical appliance, specifically showing the resistance value R_0 of a resistor, the voltage U_0 across its terminals, and the current values I_0 and I_1 measured before and after the resistor, respectively. Due to equipment limitations, the current meters recording I_0 and I_1 may have errors of up to 10% and 20%, respectively. The red-colored data represents erroneous observations, while the data in parentheses

Proceedings of the VLDB Endowment, Vol. XX, No. X ISSN 2150-8097.
doi:XX.XX/XXX.XX

Table 1: A dataset containing dirty values

	Time	I_0	U_0	R_0	I_1
t_1:	1	1	5	5	1
t_2:	2	1	5	5	1
t_3:	3	0.5	5	10	0.5
t_4:	4	0 (0.5)	5	10	0.5
t_5:	5	0.5	15	30	1 (0.5)
t_6:	6	1	20	20	1
t_7:	7	1	5	5	1
t_8:	8	1	15 (5)	5	1
t_9:	9	3	15	5	0 (3)
t_{10}:	10	0	15	5(100)	0

denotes the corresponding ground truth. Similarly, the voltmeter for U_0 and the ohmmeter for R_0 can each have a maximum error of 10%. In physics, the current through a resistor is expected to be the same at both ends, giving rise to the FD_1 $I_0 \longrightarrow I_1$. Additionally, I_0 is determined by both U_0 and R_0, leading to FD_1 $U_0R_0 \longrightarrow I_0$. When attempting to mine these data dependencies using existing AFD formulations, the selection of the threshold ϵ poses a challenge. If ϵ is chosen to be no less than 0.3, a AFD $R_0 \xrightarrow{\Psi(R_0,I_0)\leq\epsilon} I_0$ is incorrectly identified, which is not a valid dependency in reality. On the other hand, if ϵ is set below 0.3, the AFD forms of FD_1 and FD_2 are incorrectly deemed invalid. These discrepancies are caused by the mining method being misled by a portion of the noisy data. This exemplifies the limitations of using a static value for ϵ.

Addressing the limitations of the static threshold in RFD, we aim to propose **dynamic approximate functional dependency** (**DAFD**) to better support the effective expression and discovery of dependency relationships in real and noisy data. DAFD can characterize the tolerance level for low-quality data using a function based on the dependency and tailored to the data source of each attribute. It employs varying thresholds for judging dependency candidates. However, mining such dependencies is non-trivial.

Firstly, DAFD inherits the issue of a vast solution space from FD mining, posing challenges to the time and space performance of search algorithms. Secondly, DAFD inherits the difficulty of pruning from traditional AFD mining. DAFD relaxes the satisfaction degree, which prevents it from determining the invalidity of a dependency candidate based on a pair of tuples, as is the case with FD. This poses a challenge to the optimization of pruning strategies based on the instance-driven approach for the DAFD solution space.

Given the rarity of perfectly clean data for dependency mining, we typically conduct such tasks on dirty data, posing unique challenges for mining DAFDs, primarily in the following aspects:

▹ **Additional threshold calculation**: As an extension of static AFD, the threshold for each DAFD candidate must be computed during the mining process. For a dataset with m attributes, this threshold calculation incurs an additional time cost of $O(m^2 \cdot 2^m)$. Optimizing the computation of these decision thresholds to reduce this extra time overhead poses a challenge in the design of DAFD mining algorithms.

▹ **Evaluation of the mining results on dirt data**: Executing mining algorithms on inherently noisy data poses a challenge in accurately determining genuine dependencies. Ignoring data source information, such as attribute error rates, can lead to inaccurate evaluations. Therefore, evaluating DAFD mining results in the presence of dirty data is crucial for algorithm design.

Contributions. Motivated by this, we propose and address the problem of dynamic approximate functional dependency discovery, tailored for direct computation on dirty data. The contributions of this paper are summarized as follows:

(1) We introduce an extended AFD variant, namely DAFD, which is better suited to represent the quality rules underlying data containing dirty values in attributes. We establish the bijective relationship between DAFD and FD, and prove the reflexivity, augmentation, and transitivity properties of DAFD.

(2) We propose the mining algorithm named DAFDiscover. DAFDiscover can accurately capture dependencies on dirty datasets using the proposed DAFD form, which has the same time and space complexity with the classical AFD mining algorithm Tane. We further introduce an optimized DAFDiscover+, which optimizes the threshold calculation process and reduces the time complexity of threshold calculation from $O(m^2 \cdot 2^m)$ to $O(2^m)$.

(3) We introduce a method to validate the probability of the mining results' validity on dirty data, denoted as *DAFD-prob*. Based on the properties of DAFD, we theoretically prove that under the basic assumptions made in this paper, the probability of DAFD mining results being valid is not less than $\frac{1}{2} + \int_{np_0}^{n\cdot\sum_{A_i\in X\cup Y}\alpha_i} f_o(x)dx$, where n represents the total number of tuples and $X \cup Y$ encompasses all elements involved in the dependency.

(4) Comparative experiments on public datasets with 6 benchmark algorithms indicate that our proposed DAFDiscover effectively mine high-quality dependencies from dirty data. Case studies further corroborate the rationality and robustness of DAFDiscover, demonstrating its practical applicability in real-world scenarios.

Organization. In the rest of the paper, Section 2 presents related work. Section 3 introduces the definition and properties of DAFD. Section 4 introduces the DAFD mining algorithm and provides theoretical analysis and examples. Section 5 reports the experimental results, and Section 6 draws the conclusion.

2 RELATED WORK

FD discovery. Functional dependencies have long been recognized as a crucial integrity constraint in database research. Broadly, the automation of FD mining algorithms is categorized into three main approaches: schema-driven, instance-driven, and hybrid[16].

Schema-driven strategies explore the dependency candidate space, typically layer by layer, evaluating each candidate while pruning the search space. Methods employing this strategy often leverage stripped partitions to assess FD candidates and utilize Armstrong's axioms to prune the search space effectively. Representative algorithms include Tane[15], Fun[27], and FD_Mine[37], and dFD[2]. Instance-driven strategies focus on comparing attribute differences between tuple pairs to simultaneously gather evidence for multiple dependencies within the search space. Algorithms such as FDep[14], FastFD, and Dep-Miner[21]. Hybrid strategies aim to combine the strengths of both schema-driven and instance-driven methods. By switching between the two strategies, these algorithms maintain high efficiency in confirming or eliminating FD candidates. HyFD and DHyFD[33] are notable examples of hybrid algorithms.

The complete mining of FDs often entails exponential time complexity, rendering it impractical for certain real-world applications. Researchers have recently explored approximate and dynamic FD mining as potential solutions. Approximate FD mining sacrifices some accuracy in minimality to improve efficiency, e.g., Aid-FD[5] and EulerFD[19]. Dynamic FD mining, meanwhile, focuses on efficiently updating FD sets in response to changes (insertions, deletions) in existing datasets, e.g., DhsFD[36].

Approximate FD discovery. Tane[15] also proposes AFD discovery solution, which calculates the minimum number of tuples to be deleted using stripped partitions. It employs a breadth-first strategy, similar to that used in mining exact FDs, to traverse the search space effectively. Pyro[18] constructs a left-hand side search space for each attribute, enabling the discovery of all AFDs in the dataset. Pyro utilizes two phases, ascend and trickle-down, to traverse the search space efficiently. Both phases incorporate a sampling-based error estimation strategy to reduce traversal time. In addition, Cords[17], utilizes a sampling strategy for mining soft functional dependencies (SFD). Meanwhile, a method for computing partial functional dependencies (PFD) is introduced in [32]. Furthermore, [6] presents Domino, which focuses on mining relaxed functional dependencies (RFD) with a more lenient comparison approach.

To accommodate dependency mining tasks in scenarios involving low-quality data, some algorithms adopt non-RFD mining approaches. [22] introduces RFI, a top-k mining technique based on a branch-and-bound strategy. It utilizes a mutual information-based metric to characterize dependency relationships between attributes. [38] proposes FDx, which targets noisy data, by framing the FD mining problem as a structure learning task.

In summary, existing AFD semantics often struggle to adapt to real-world data inherently containing noise or erroneous data. Furthermore, current mining algorithms lack the ability to adjust relaxation thresholds based on data's characteristics. As obtaining sufficient and entirely clean data is often challenging in practice, there is an urgent need for methods that can effectively mine dependencies directly from dirty data. Our work in this paper provides a reliable solution to addressing the aforementioned challenges.

3 OVERVIEW OF DAFD

3.1 Preliminaries

Let $\mathcal{R}$ be a relation and $Attr(\mathcal{R}) = (A_1, ..., A_m)$ represents the set of attributes in $\mathcal{R}$. $\mathcal{I}$ is an instance of relation $\mathcal{R}$ containing n tuples, each of which belongs to the domain $Dom(A_1) \times \cdots \times Dom(A_m)$. $Dom(A)$ represents the domain of attribute A. $t[A]$ records the data value of tuple t on attribute A. One **function dependency (FD)** on $\mathcal{R}$ is defined as $\varphi : \mathcal{R}(X \rightarrow Y)$, where $X, Y \subseteq Attr(\mathcal{R})$. φ is regarded to be in normal form iff $X \cap Y = \emptyset$, and Y only consists of a single attribute A. X and Y are denoted as the left-hand side (LHS) and the right-hand side (RHS) of φ, respectively.

One **approximate function dependency (AFD)** on $\mathcal{R}$ is defined as $\varphi : X \xrightarrow{\Psi \leq \epsilon} Y$, where $X, Y \in Attr(\mathcal{R})$. Ψ is a coverage measure defined on $Dom(\mathcal{R})$ quantifying the amount of tuples violating or satisfying φ, and ϵ is a threshold indicating the upper bound for the result of the coverage measure.

We propose the DAFD form in Definition 3.1. It achieves dynamic relaxation from the perspective of satisfaction degree, aiming to better support the computational analysis of dependency patterns in low-quality data containing dirty values.

Definition 3.1. **Dynamic Approximate Functional Dependencies (DAFD)**. Given $\mathcal{I}_D$ as a low-quality data instance containing dirty values from $\mathcal{R}$, let $X \xrightarrow{\Psi(X,Y) \leq \alpha(X,Y)} Y$ represent a DAFD on $\mathcal{I}_D$. This DAFD holds true if at least one of the following conditions is satisfied: (1) When $|Y| = 1$ and $\Psi(X,Y) \leq \alpha(X,Y)$, or (2) When $|Y| > 1$ and $\forall y \in Y$, $\Psi(X,y) \leq \alpha(X,y)$, where $\Psi(X,Y) = \frac{\min\left\{|t| \mid t \subseteq \mathcal{I}_D \text{ and } X \rightarrow Y \text{ holds in } \mathcal{I}_D \backslash t\right\}}{|\mathcal{I}_D|}$, representing a measure of dependency satisfaction. Function $\alpha(X,Y) = \sum_{A_i \in X \cup Y} a_i$ represents the error threshold, where a_i is the upper bound on the proportion of erroneous data for the corresponding attribute in data $\mathcal{I}_D$. This reflects the inherent nature of the data source itself.

3.2 The nature of DAFD

3.2.1 The fundamental assumption regarding datasets containing dirty values. Regarding our research subject, the dataset $\mathcal{I}_D$ containing dirty values, we make the following assumptions: (a) The error probability of each attribute's data source is relatively low, and there exists an upper bound on the proportion of erroneous data. (b) If a FD candidate does not hold in a particular low-quality dataset, the proportion of tuples violating this FD candidate significantly exceeds (*e.g.,* by more than 10 times) the proportion of tuples containing erroneous attribute values. (c) Whether the value of each tuple under the same attribute is erroneous is mutually independent, and (d) Within the same tuple, the erroneousness of any set of attribute values does not decrease the probability of another attribute value being erroneous.

We emphasize that these assumptions align with real-world scenarios. Our analysis is based on the existence of an upper bound on the proportion of erroneous data within each tuple of the dataset, and we provide our theoretical analysis in detail in Section 4,

3.2.2 DAFD vs. FD. For the DAFD described in Definition 3.1, when we specify $\alpha(X,Y) = \sum_{A_i \in X \cup Y} a_i$, the relationship between DAFD and its corresponding FD is observed in Theorem 3.2.

Theorem 3.2. *Let $\mathcal{I}$ be a completely clean dataset on the relation schema $\mathcal{R}$, and $\mathcal{I}_d$ be a low-quality dataset on $\mathcal{R}$, a DAFD on $\mathcal{I}_d$ holds if and only if the corresponding isomorphic FD $\mathcal{I}$ holds.*

Herein, an FD is isomorphic w.r.t a DAFD if and only if the terms (i.e., *attributes) in their LHS and RHS are identical.*

Proof. Given a DAFD φ, (i) when $|RHS(\varphi)| = 1$, if the DAFD does not hold on $\mathcal{I}_D$, even if all tuples with erroneous data in $\mathcal{I}_D$ violate the DAFD, there must still be some tuples with entirely correct data that violate the DAFD (otherwise, the DAFD would hold on $\mathcal{I}_D$). Consequently, these tuples violate the corresponding FD on $\mathcal{I}$, and thus the FD on $\mathcal{I}$ does not hold. If the DAFD holds on $\mathcal{I}_D$, assume that the corresponding FD does not hold on $\mathcal{I}$. Then, at most $\Sigma_{A_i \in X \cup Y} 2a_i n$ tuples in $\mathcal{I}$ violate the FD. This means that the number of tuples violating the dependency cannot satisfy the condition of being significantly larger than the number of tuples with erroneous data, contradicting the assumption (b). Therefore, the FD must hold on $\mathcal{I}$.

(ii) When $|RHS(\varphi)| > 1$, if the DAFD holds on $\mathcal{I}_D$, then $\forall y \in RHS(\varphi)$, the DAFD formed by the LHS and y holds. By the argument in (i), this implies that for all $y \in RHS(\varphi)$, the FD having the same shape with DAFD formed by the LHS and y holds on $\mathcal{I}$. Therefore, the FD holds on $\mathcal{I}$. If the DAFD does not hold on $\mathcal{I}_D$, then there exists a $y \in RHS(\varphi)$ such that the DAFD formed by the LHS and y does not hold. By (i), this implies that $\exists y \in RHS(\varphi)$ such that the FD formed by the LHS and y does not hold on $\mathcal{I}$. Since for FDs,

$X \rightarrow Y$ holds if and only if for all $y \in RHS(\varphi)$, $LHS \rightarrow y$ holds, it follows that $X \rightarrow Y$ does not hold on $\mathcal{I}$.

Thus, an FD holds on $\mathcal{I}$ if and only if the corresponding DAFD holds on $\mathcal{I}_D$. □

3.2.3 DAFD reasoning. Theorem 3.3 introduces the inference system of DAFD, which is crucial for the mining of DAFD from data.

Theorem 3.3. *DAFD satisfies the Armstrong axiom system, that is, DAFD satisfies the law of reflexive, augmentation, and transitive.*

Proof. **Reflexive**. Based on the property of FD [15], it is always true that $X \rightarrow X$. Furthermore, according to Theorem 3.2, $X \xrightarrow{\Psi(X,X)\leq\alpha(X,X)} X$ holds true consistently. **Augmentation**. If $X \xrightarrow{\Psi(X,Y)\leq\alpha(X,Y)} Y$ holds true, then according to Theorem 3.2, it follows that $(X \rightarrow Y)$ is valid. Furthermore, based on the properties of FD, it has $XA \rightarrow YA$ $(\forall A \in Attr(\mathcal{R}))$. Consequently, by Theorem 3.2, $\forall A \in Attr(\mathcal{R})$, $XA \xrightarrow{\Psi(XA,YA)\leq\alpha(XA,YA)} YA$ is also established. **Transitive**. Let $X \xrightarrow{\Psi(X,Y)\leq\alpha(X,Y)} Y$ and $Y \xrightarrow{\Psi(Y,Z)\leq\alpha(Y,Z)} Z$ be two DAFDs that hold true on $\mathcal{I}_D$. Accordingly to Theorem 3.2, $X \rightarrow Y$ and $Y \rightarrow Z$ are valid on the corresponding clean dataset $\mathcal{I}$. Due to the transitivity property of FD, $X \rightarrow Z$ is also valid on $\mathcal{I}$. Consequently, $X \xrightarrow{\Psi(X,Z)\leq\alpha(X,Z)} Z$ is established on $\mathcal{I}_D$. □

3.2.4 A comparative analysis of the properties of DAFD and other FD variations. Table 2 summarizes FD and several of its variants, along with their typical mining algorithms. Most data dependencies require strict equality in attribute values. AFD, PFD, and SFD allow for some relaxation in satisfaction, exhibiting a certain degree of tolerance and adaptability to poor-quality data. However, they only support a fixed level of tolerance. DAFD, on the other hand, with its support for dynamic ϵ, can better adapt to computations in real-world scenarios. Furthermore, our characterization of the upper bound on attribute error rates in $\alpha(X, Y)$ also reflects the description of data source information, thus enhancing the expressive power of DAFD semantics. Although traditional AFD does not satisfy the Armstrong axiom system, DAFD does, similar to FD. This provides a significant advantage for the efficient mining of DAFD.

4 DISCOVERING DAFD

Section 4.1 presents the mining problem of DAFD, while Section 4.2 details the specific steps of the algorithm. Section 4.3 provides a theoretical analysis of the results, and Section 4.4 discusses the algorithm's time and space complexity, as well as its robustness.

4.1 The DAFD discovery problem

We first formalize the problem of mining DAFDs in Problem 1.

Problem 1. *Given a dirty dataset $\mathcal{I}_D$ of schema $\mathcal{R}(A_1, ..., A_m)$ and known upper bounds on the proportion of erroneous data within each attribute, denoted by $a_i, i \in [1, m]$, the DAFD mining problem aims to find all valid, minimal and non-trivial DAFDs from $\mathcal{I}_D$.*

*Here, (i) one DAFD φ is identified to be **valid** w.r.t $\mathcal{I}_D$ iff $\forall \varphi \in \Sigma_{\mathsf{DAFD}}$, φ hold true in $\mathcal{I}_D$. (ii) One DAFD φ: $X \xrightarrow{\Psi(X,Y)\leq\alpha(X,Y)} Y$ is **minimal** iff $\forall X' \subseteq X$, $X' \xrightarrow{\Psi(X',Y)\leq\alpha(X',Y)} Y$ is not valid, and (iii) one DAFD φ: $X \xrightarrow{\Psi(X,Y)\leq\alpha(X,Y)} Y$ is **non-trivial** iff $Y \notin X$.*

According to Theorem 3.2, there exists a one-to-one correspondence between DAFD on dirty dataset $\mathcal{I}_D$ and FD on the corresponding clean dataset $\mathcal{I}$. Therefore, Theorem 4.1 holds true for the dependency discovery problem.

Theorem 4.1. *Let Σ_{DAFD} be the mining result for DAFD on a dirty dataset $\mathcal{I}_D$, and Σ_{FD} is the set of FDs that are isomorphic to the DAFDs in Σ_{DAFD}, then Σ_{FD} represents the mining result for FDs on the corresponding clean dataset $\mathcal{I}$ to which $\mathcal{I}_D$ belongs.*

Complexity discussion. The FD mining problem has been proven to be bounded by $O(n^2(\frac{m}{2})^2 2^m)$ in time, where n represents the number of tuples in the dataset and m denotes the number of attributes [20]. Note that DAFD mining incurs additional computational overhead for calculating dynamic thresholds, the DAFD mining can traverse the search space analogously to FD mining, with each dependency requiring $O(n)$ time for evaluation. The total time for DAFD mining includes both the time for dependency evaluation and the extra time for computing thresholds, amounting to $O(n^3(\frac{m}{2})^2 2^m + T(m, n))$, where $T(m, n)$ represents the cumulative time for computing thresholds throughout the mining process.

Similar to FD and AFD, DAFD discovery faces the challenge of effectively pruning the search space of $O(m \cdot 2^m)$. However, since DAFD relax the satisfaction criteria by tolerating minor violations (i.e., dirty values) in data, it becomes challenging to simultaneously evaluate and prune multiple dependency candidates through tuple pair comparisons. Furthermore, each DAFD candidate requires the computation of its threshold, adding $T(m, n)$ as additional time in mining. Under a brute-force approach, $T(m, n) = O(m^2 2^m)$, highlighting the importance of effective pruning strategies for reducing $T(m, n)$ as a critical aspect.

4.2 DAFDDiscover algorithm

4.2.1 The holistic design of DAFDDiscover. Insights into DAFD mining techniques can be drawn from the properties in Section 3.2.4. DAFDs are semantically similar to AFDs, with the difference being that DAFDs replace ϵ in AFDs with a function of the dependency relationship. This suggests that DAFD mining can be developed by appropriately extending SOTA AFD mining algorithms. Intuitively, for any given AFD mining algorithm, converting ϵ into the function $\alpha(X, Y) = \sum_{A_i \in X \cup Y} a_i$ can enable the discovery for DAFDs. However, it is worth noting that AFDs do not satisfy the Armstrong axiom system, while DAFDs do satisfy the Armstrong axiom system. This allows us to adopt more aggressive pruning strategies, for faster pruning of the search space. By leveraging this, we can potentially enhance the efficiency and effectiveness of DAFD mining algorithms.

Accordingly, the DAFD mining consists of two pivotal steps: (i) Substituting the hyperparameter ϵ of the AFD mining algorithm with function $\alpha(X, Y) = \sum_{A_i \in X \cup Y} a_i$, and (ii) Employing pruning strategies, which include those based on the Armstrong axiom system, to efficiently prune the search space.

Next, we present our mining algorithm in Section 4.2.2, followed by further optimization algorithms in Section 4.2.3.

4.2.2 The specific steps of DAFDDiscover. In a review of the prevalent AFD mining algorithms, Tane [15] stands out as a schema-driven approach that employs a breadth-first strategy to traverse the

Table 2: Summary of DAFD and other potential FD variations for expressing data dependencies on dirty data.

Data dependencies			Definition	Violation tolerance	Reasoning	Representative methods		Time complexity	Space complexity	Results
FD	no tolerance	FD	X→Y	No	Armstrong rules	Exact Discovery	TANE[15]	$O(2^m(n+m^{2.5}))$	$O(\frac{(n+m)2^m}{\sqrt{m}})$	All FDs
							FastFD[35]	-	$O(\lvert Difference\_set \rvert m)$	All FDs
							HYFD[28]	$O(mn^2+m^2 2^m)$	-	All FDs
						Approximate Discovery	EulerFD[19]	almost linearly with row expansion	-	All FDs
							AID-FD[5]	-	-	All FDs
	extent of satisfaction	AFD	$X \xrightarrow{\frac{\lvert\mathcal{I}_D\rvert-\max\left\{\lvert t\rvert \mid t\subseteq\mathcal{I}_D \text{ and } X\to Y \text{ holds in } t\right\}}{\lvert\mathcal{I}_D\rvert}\leq\epsilon} Y$	Fixed	$X \to Y \Rightarrow XA \to Y$	TANE[15]		$O(2^m(mn+m^{2.5}))$	$O(\frac{(n+m)2^m}{\sqrt{m}})$	All AFDs
			$X \xrightarrow{\frac{\lvert\{(t_1,t_2)\in\mathcal{I}_D{}^2 \mid t_1[X]=t_2[X]\wedge t_1[Y]\neq t_2[Y]\}\rvert}{\lvert\mathcal{I}_D\rvert^2-\lvert\mathcal{I}_D\rvert}\leq\epsilon} Y$	Fixed	$X \to Y \Rightarrow XA \to Y$	PYRO[18]		-	-	All AFDs
		PFD[32]	$X \xrightarrow{\frac{\sum_{x\in dom(X)} P(X\to Y,V_x)}{\lvert dom(X)\rvert}\geq 1-\epsilon} Y$	Fixed	unknown	extension of TANE[32]		-	-	-
		SFD[17]	$X \xrightarrow{\frac{\lvert dom(X)\rvert_{\mathcal{I}_D}}{\lvert dom(X,Y)\rvert_{\mathcal{I}_D}}\geq 1-\epsilon} Y$	Fixed	$X \to Y \Rightarrow XA \to Y$	CORDS[17]		-	-	SFDs with single attribute in LHS
		DAFD	$X \xrightarrow{\frac{\min\left\{\lvert t\rvert \mid t\subseteq\mathcal{I}_D \text{ and } X\to Y \text{ holds in } \mathcal{I}_D\setminus t\right\}}{\lvert\mathcal{I}_D\rvert}\leq\alpha(X,Y)} Y$	Dynamic	Armstrong rules	DAFDiscover DAFDiscover+		$O(2^m(nm+m^{2.5}))$	$O(\frac{(n+m)2^m}{\sqrt{m}})$	All DAFDs
Mutual information		MI score	various definitions based on mutual information	Fixed	varying with specific definitions	RFI[22]		-	-	Top-k FD discovery
						SMI[29]		-	-	
Structure learning			$X \to Y$, with the set of attributes X in $\mathcal{R}$ that correspond to non-zero entries in autoregression matrix[38]	based on the process of learning	-	FDX[38]		quadratic complexity with respect to the number of columns	-	One FD for each RHS attribute

search space of AFD candidates, ultimately uncovering all AFDs within a dataset. During the exploration of the AFD candidate space, Tane incorporates three pruning strategies to optimize its performance: (i) If $X \xrightarrow{\Psi(X,A)\leq\epsilon} A$ holds, then $\forall B \in Attr(\mathcal{R})$, $XB \xrightarrow{\Psi(XB,A)\leq\epsilon} A$ holds. (ii) If $X \xrightarrow{\Psi(X,A)\leq\epsilon} A$ is a valid FD, then $\forall B \in Attr(\mathcal{R})$, $XB \xrightarrow{\Psi(XB,A)\leq\epsilon} A$ cannot be a minimal AFD. (iii) If X is a superkey i.e., $\forall t_1, t_2 \in \mathcal{I}, t_1[X] \neq t_2[X]$, then $\forall B \in Attr(\mathcal{R})$, $X \xrightarrow{\Psi(X,B)\leq\epsilon} B$ holds. The pruning strategies (i) and (ii) in Tane are facilitated by the utilization of the $C^+(X)$ set, which is defined as $C^+(X) = \{A \in Attr(\mathcal{R}) | \forall B \in X : X\{A,B\} \to \{B\} \text{is not valid}\}$. X can be pruned when $C^+(X) = \emptyset$.

Therefore, the initial intuition is to extend Tane for mining DAFDs. As mentioned above, we can modify ϵ in AFD to $\alpha(X,Y) = \sum_{A_i\in X\cup Y} a_i$. Subsequently, we need to review and adapt the pruning strategies employed in Tane. Regarding the pruning based on $C^+(X)$ and the Armstrong axiom system, since DAFDs satisfy the Armstrong axiom system, we can revert to the pruning strategies originally employed by TANE for FD mining.

However, it is important to emphasize that the key pruning strategy proposed in Tane is not suitable for mining DAFDs! This is because it can lead to the omission of dependency relationships during the mining process. When an attribute set X is determined to be a key, the key pruning strategy in Tane removes X from L_l, resulting in the pruning of all sets $X \cup A$ in L_{l+1}. Specifically, all DAFD of the form $X \xrightarrow{\Psi\leq\alpha} A$ and $(X \cup \{A\}) - \{B\} \xrightarrow{\Psi\leq\alpha} B$ are pruned. However, in the context of DAFD mining, when X is identified as a key, it only ensures that all DAFDs of the form $X \xrightarrow{\Psi\leq\alpha} A$ are valid and have been discovered in L_l. It does not guarantee the validity of DAFD of the form $(X\cup\{A\})-\{B\} \xrightarrow{\Psi\leq\alpha} B$, nor does it ensure that the valid ones can be discovered in L_l on attribute sets other than the removed $X \cup A$. As a result, applying Tane's pruning strategy in DAFD mining leads to the omission of dependency relationships during the mining process.

Fortunately, we have discovered that in DAFD mining process, if an attribute set X is determined to be a key, then all DAFD of the form $X \xrightarrow{\Psi\leq\alpha} A$ are valid. Consequently, if A is present in $C^+(X \cup A)$, we can remove A and any attributes in $C^+(X \cup A)$ that are not in X. Furthermore, if an attribute set Y in L_{l+1} satisfies the condition that $\forall B \in Attr(\mathcal{R})$, $Y - \{B\}$ is a key, then $C^+(Y) = \emptyset$, and we can prune the attribute set Y.

Based on the above, we propose a DAFD mining algorithm with a novel key pruning strategy tailored to the requirements of DAFD mining. Specifically, when X in L_l is identified as a key, we mark X as a key and retain the set. When generating L_{l+1}, for each newly generated set Y, we check if for all $B \in Attr(\mathcal{R})$, if $Y \setminus \{B\}$ is a key, we remove B and any attributes in $C^+(Y)$ that are not in X. If $C^+(Y) = \emptyset$, indicating that all dependencies on Y have been output using key attribute sets in L_l, we can prune Y from L_{l+1}.

The overall process is outlined in Algorithm 1. It initializes the attribute sets with 0 and 1 attributes, along with their corresponding C^+ collections, grouping sets with the same number of attributes into the same level. Subsequently, function `Compute_dependencies`(L_l) (Algo 2) is invoked to calculate the DAFDs that hold true for each attribute set in every level and update C^+ collections accordingly. After that, the `Prune`(L_l) function (Algo 3) is utilized to prune attribute sets with empty C^+s, filter out the keys from the attribute sets, and modify the C^+s using a key pruning strategy. Based on this, the `Generate_next_level`(L_l) function (Algo 4) is employed to generate the attribute sets for the next level from the current level's sets and proceed with the mining process for the subsequent levels until no attribute sets are generated for the next level.

`Compute_Dependencies` (Algorithm 2) operates on each attribute set X in the current level. It examines every potential DAFD candidate of the form $X \setminus \{A\} \xrightarrow{\Psi(X\setminus\{A\},A)\leq\sum_{A_i\in X} a_i} A$ for X. It then outputs the valid DAFDs and updates $C^+(X)$ for X accordingly.

The `Prune` function (Algorithm 3), removes attribute sets from L_l that have an empty C^+ set. It identifies and labels attribute sets determined to be keys and outputs the minimal and non-trivial

Algorithm 1: DAFDiscover

```
Input  : dirty data $\mathcal{I}_D$, the upper limit on the proportion of
         erroneous data $a_i$ for each attribute $A_i$.
Output : the set of DAFDs, $\Sigma_{\mathrm{DAFD}}$
1 $L_0 \leftarrow \{\emptyset\}, C^+(\emptyset) \leftarrow Attr(\mathcal{R}), L_1 \leftarrow \{\{A\} | A \in Attr(\mathcal{R})\}$;
2 $l \leftarrow 1$;
3 foreach $X \in L_1$ do
4     $C^+(X) \leftarrow \bigcap_{A \in X} C^+(X \setminus \{A\})$;
5 while $L_1 \neq \emptyset$ do
6     Compute_dependencies($L_l$);
7     Prune($L_l$);
8     $L_{l+1} \leftarrow$ Generate_next_level($L_l$), $l \leftarrow l + 1$;
```

Algorithm 2: Compute_dependencies(L_l)

```
Input  : a set of attribute sets $L_l$ with $l$ attributes
Output : the remaining DAFDs on $L_l$
1 foreach $X \in L_l$ do
2     foreach $A \in X \cap C^+(X)$ do
3         if $e(X \setminus \{A\} \to A) \leq \sum_{A_i \in X} a_i$ then
4             return $X \setminus \{A\} \xrightarrow{\Psi(X \setminus \{A\}, A) \leq \sum_{A_i \in X} a_i} A$;
5             remove $A$ from $C^+(X)$;
6             remove all $B$ in $R \setminus X$ from $C^+(X)$;
```

DAFDs that can be generated. The key labels of attributes are then utilized in the generation of attribute sets for the subsequent level.

Algorithm 3: Prune(L_l)

```
Input  : a set of attribute sets $L_l$ with $l$ attributes
Output : the pruned $L_l$
1 foreach $X \in L_l$ do
2     if $C^+(X) = \emptyset$ then
3         delete $X$ from $L_l$;
4     if $X$ is an (super) key then
5         foreach $A \in C^+(X) \setminus X$ do
6             if $A \in \bigcap_{B \in X} C^+(X \cup \{A\} \setminus \{B\})$ then
7                 return $X \to A$;
8             Record $X$ as a key;
```

The Generate_next_level(L_l) function (Algo 4), generates attribute sets with l+1 attributes and their corresponding C^+ set based on L_l. It performs validity checks on the generated attribute sets and places those that pass the checks into L_{l+1}. Within this process, Prefix_Blocks divides L_l into multiple blocks (prefix blocks). For each attribute set in L_l, after sorting its attributes, if two sets have the same number of attributes and differ only in their final sorted attribute, they are considered to be in the same prefix block[24].

The determination process for X_is_valid, as outlined in Algorithm 5, involves assessing the validity of the input X. If $\forall x \in X$, $\exists aset \in L_l$ such that $aset = X \setminus \{x\}$, then X is considered a valid attribute set of length $l + 1$. During this process, it checks the key labels of the attribute set *aset* and performs key pruning on X accordingly. If X is ultimately determined to be valid and its $C^+(X)$ is not empty, it concludes that X can be included in L_{l+1}.

Algorithm 4: Generate_next_level(L_l)

```
Input  : a set of attribute sets $L_l$ with $l$ attributes
Output : the next level of attribute sets, $L_{l+1}$.
1 $L_{l+1} \leftarrow \emptyset$;
2 foreach $K \in$ Prefix_Blocks($L_l$) do
3     foreach $\{Y, Z\} \subseteq K, (Y \neq Z)$ do
4         $X \leftarrow Y \cup Z, C^+(X) \leftarrow \bigcap_{A \in X} C^+(X \setminus \{A\})$;
5         if X_is_valid($L_l, X$) then
6             $L_{l+1} \leftarrow L_{l+1} \cup \{X\}$;
7 return $L_{l+1}$;
```

Algorithm 5: X_is_valid(L_l, X)

```
Input  : a set of attribute sets $L_l$ with $l$ attributes, and a candidate
         attribute set $X$ for validation.
Output : a boolean flag indicating true or false
1  for $x \in X$ do
2      $check \leftarrow 0$;
3      for $aset \in L$ do
4          if $aset = X \setminus \{x\}$ then
5              $check \leftarrow 1$;
6              if $aset$ is key then
7                  remove $A$ from $C^+(X)$;
8                  remove all $B$ in $Attr(\mathcal{R}) \setminus X$ from $C^+(X)$;
9              Break;
10     if $check = 0$ then
11         return False;
12 if $C^+(X) = \emptyset$ then
13     return False;
14 return True;
```

4.2.3 *Optimization of DAFDiscover.* We emphasize that there are still possibilities for further improvements to DAFDiscover. We have identified two key areas for optimization. (*i*) DAFDiscover requires the calculation of the decision threshold $\sum_{A_i \in X \cup Y} a_i$ for each DAFD candidate, which involves redundant computations. (*ii*) the key pruning strategy employed by DAFDiscover can be made more aggressive. Addressing these two issues leads to the optimized version of DAFDiscover, henceforth referred to as DAFDiscover+.

(1) **Reducing redundant threshold calculations**. DAFDiscover computes the decision threshold $\sum_{A_i \in X \cup Y} a_i$ for each unpruned DAFD candidate, leading to $O(m^2 2^m)$ in time in the overall mining. However, there is redundancy in these summations. For all $A \in X$, computing the decision threshold for $X \setminus \{A\} \to A$ using $\sum_{A_i \in X \setminus \{A\} \cup A} a_i$ will yield the same result, namely $\sum_{A_i \in X} a_i$. Since DAFDiscover traverses all DAFD candidates under X in the pattern $X \setminus \{A\} \to A$ for each X, we can optimize by computing the corresponding decision threshold only once when generating X and reusing it for all DAFD candidates under X. This optimization reduces redundant threshold calculations, resulting in a reduced time complexity of $O(m^2 2^m)$ to $O(m 2^m)$, which is implemented in Compute_dependencies, and requires only a minor modification to the first four lines in Algorithm 2 as follows. That is, when calculating the DAFDs for any attribute set X in L_l, we first compute

```
1': foreach $X \in L_l$ do
2':     $\epsilon_X = \sum_{A_i \in X} a_i$;
3':     foreach $A \in X \cap C^+(X)$ do
4':         if $e(X \setminus \{A\} \to A) \le \epsilon_X$ then
5':             return $X \setminus \{A\} \xrightarrow{\Psi(X\setminus\{A\},A)\le\epsilon_X} A$;
```

the corresponding threshold ϵ_X for that attribute set and then use this ϵ_X directly in subsequent DAFD calculations for X.

Further, we optimize the efficiency of the `Generate_next_level`(L_l). It is worth noting that each attribute set at level L_{l+1} is derived by appending an attribute to a specific attribute set at level L_l. Correspondingly, the threshold $\sum_{A_i \in X} a_i$ for X at L_{l+1} can be obtained by adding the tolerance limit of the appended attribute to $\sum_{A_i \in Y} a_i$ of its prefix Y at L_l. With this optimization, the computing of $\sum_{A_i \in X} a_i$ for each set X becomes $O(1)$, and the overall time complexity for computing thresholds in the entire algorithm reduces to $O(2^m)$.

This optimization shifts the computation of $\sum_{A_i \in Y} a_i$ from `Compute_dependencies` to `Generate_next_level`(L_l). Consequently, for function `Compute_dependencies`, by directly utilizing the precomputed threshold ϵ_X, the modification in Algorithm 2 (line 3) is:

```
3':     if $e(X \setminus \{A\} \to A) \le \epsilon_X$ then
```

Accordingly, in Algorithm 4, the fifth line is modified as follows:

```
5':   if X_is_valid$(L_l, X)$ then
6':       $\epsilon_X \leftarrow \epsilon_Y + \epsilon_{X\setminus Y}$;
```

While generating X in L_{l+1}, ϵ_X is produced by leveraging the threshold ϵ_Y of its prefix attribute set Y and the threshold related to the additional part of X compared to Y, denoted as $X \setminus Y$ (which, due to the properties of `Prefix_Blocks`, should contain only one attribute). Since `Prefix_Blocks` is maintained beforehand, the sets Y and Z in line 3 of Algorithm 4 differ by only one attribute, and when the attributes in the sets are sorted, the differing attribute is always the last one. This allows us to compute the threshold for a new attribute set using the threshold of a known set, with $O(1)$ time for a single ϵ_X calculation.

(2) **More Aggressive Pruning Strategy**. DAFDiscover only perform key pruning when an attribute set strictly qualifies as a superkey. Further, when an attribute set X is an *approximate superkey*, meaning it can become a superkey by removing tuples that constitute no more than the proportion $\sum_{A_i \in X} a_i$, the proportion of violating tuples for X as a DAFD candidate with LHS will not exceed $\sum_{A_i \in X} a_i$. Consequently, it will not exceed $\sum_{A_i \in X \cup Y^*} a_i$, where $Y^* \in Attr(\mathcal{R})$ represents the RHS of the DAFD candidate with X as the LHS. This ensures that X, as the LHS of a DAFD candidate, is guaranteed to form a valid DAFD. Therefore, it is feasible to perform key pruning on approximate superkeys, enabling a more aggressive pruning strategy than that used by DAFDiscover.

This optimization is implemented within the `Prune`(L_l) procedure by relaxing the condition in line 4 of Algorithm 3 from strictly checking for superkeys to allowing for approximate superkeys.

```
4':   if $X$ is an approximate_(super)key then
```

The determination of whether X is an approximate super key can be achieved by calculating the proportion of the minimum number of tuples that need to be removed from X's stripped partition[15] to make it a key, relative to the total number of tuples. This proportion is then compared to $\sum_{A_i \in X} a_i$ to assess if it exceeds the threshold.

From the above, we obtain the enhanced DAFDiscover algorithm, named DAFDiscover+. The overall flow of DAFDiscover+ remains largely similar to the original DAFDiscover. The only modification required is to update the initialization process in Algorithm 1 line 1 as follows:

```
1': $L_0 \leftarrow \{(\emptyset, 0\}, C^+(\emptyset) \leftarrow Attr(\mathcal{R}), L_1 \leftarrow \{(A_i, a_i\}|A_i \in Attr(\mathcal{R})\}$;
```

Example 4.2. Fig. 1 illustrates the mining process for the data in Example 1.1. Except $I_0 \to I_1$, we focus on mining the data under attributes Time (T), I_0, U_0, and R_0. DAFDiscover+ initializes L_0, L_1, l, and the set C^+ and performs a layer-by-layer traversal to mine DAFDs. During each traversal of a layer, DAFDiscover+ computes dependencies in the form $X \setminus \{A\} \to A$ for each attribute set in L_l (i.e., attribute sets with l attributes at level l) using `Compute_dependencies`. It then prunes the C^+ set of each attribute set based on the discovered dependencies. Subsequently, `Prune` removes attribute sets with empty C^+s from the current L_l (shown in green in Fig. 1) and identifies and marks key attribute sets (in red). The corresponding DAFDs for the key attribute sets are output.

Next, `Generate_next_level` generates attribute sets with $l+1$ attributes along with their corresponding C^+ sets. It performs key pruning on the generated C^+ sets based on the marked key attribute sets in L_l and adds attribute sets with non-empty C^+ sets to L_{l+1} (shown in black). Finally, it updates l and passes L_{l+1} to the next iteration of `Compute_dependencies` for mining the next layer of attribute sets. The iteration continues until L_l becomes empty, indicating the completion of the mining process.

4.3 Theoretical analysis of mining results

Theorem 4.3 outlines the quality of the mining results obtained by DAFDiscover+ utilizes a more robust pruning optimization strategy, is capable of achieving the same outcomes as DAFDiscover.

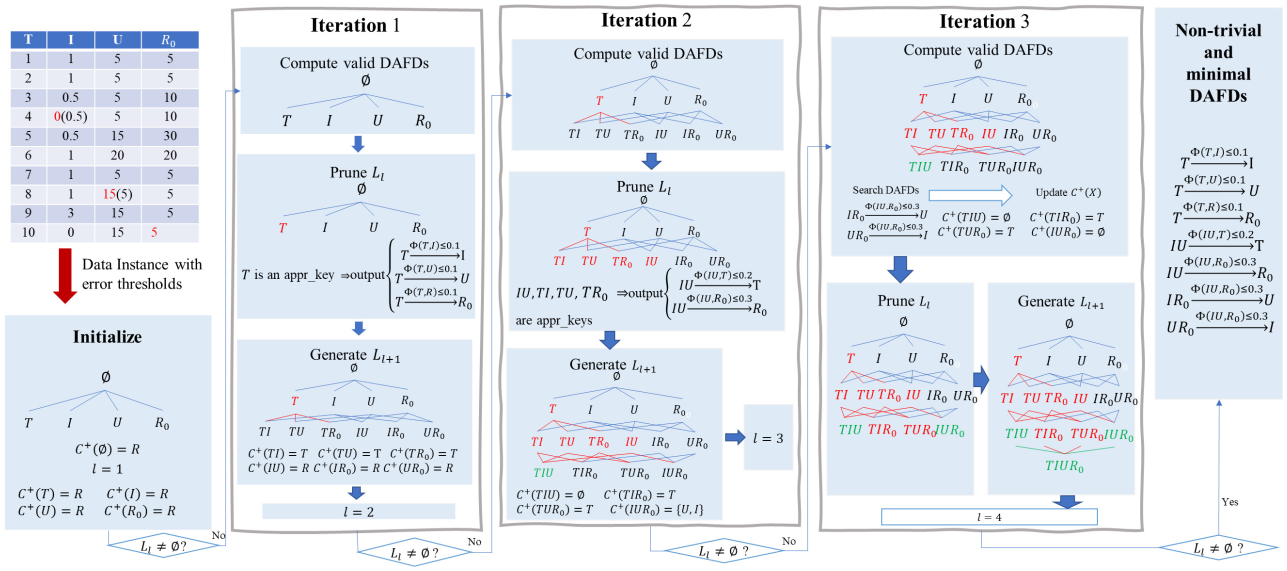
Theorem 4.3. *For the same dataset I_D, the mining results of DAFDiscover+ are identical to those of the DAFDiscover.*

Proof. Denoting the sets mined by DAFDiscover and DAFDiscover+ as Σ and Σ^+, respectively. Compared to DAFDiscover, DAFDiscover+ introduces two optimizations: an improved threshold calculation process and the relaxation of key pruning to approximate key pruning. In the optimization of the threshold calculation, redundant computations are eliminated. However, the actual values involved in determining the thresholds for each DAFD candidate remain unchanged. Therefore, this optimization does not lead to differences between Σ and Σ^+.

In the optimization of approximate key pruning, as it is a relaxation of key pruning, the dependencies output in approximate key pruning include those output during the key pruning process. Therefore, it suffices to prove that the dependencies pruned in relaxed key pruning are all output by DAFDiscover. If X is determined to be an approximate key, at most $\sum_{A_i \in X} a_i$ tuples can be removed, and the attribute set can then be considered a key. Consequently, for any DAFD candidate $X \to A$ with X as the LHS, it has $\Psi(X, A) \le \sum_{A_i \in X} a_i \le \sum_{A_i \in X \cup A} a_i$), meaning $X \xrightarrow{\Psi(X,A)\le\sum_{i\in X\cup A} a_i} A$. Thus, every DAFD candidate pruned satisfies the DAFD definition and would be output in DAFDiscover through key pruning or `Compute_dependencies`. Therefore, this optimization does not cause differences between Σ and Σ^+.

In summary, the optimizations employed in DAFDiscover+ do not result in differences between Σ and Σ^+, hence the mining results of both algorithms are identical. □

Since the mining results of these two algorithms are identical, we analyze the properties of the mining results obtained by DAFDiscover. Firstly, we prove the properties of the mining results Σ in Theorem 4.4.



T I U R_0
1 1 5 5
2 1 5 5
3 0.5 5 10
4 0(0.5) 5 10
5 0.5 15 30
6 1 20 20
7 1 5 5
8 1 15(5) 5
9 3 15 5
10 0 15 5
Data Instance with error thresholds
Initialize
$C^+(\emptyset) = R$
$l = 1$
$C^+(T) = R$ $C^+(I) = R$
$C^+(U) = R$ $C^+(R_0) = R$
Iteration 1
Compute valid DAFDs
Prune L_l
T is an appr_key ⇒output
$T \xrightarrow{\Phi(T,I)\le 0.1} I$
$T \xrightarrow{\Phi(T,U)\le 0.1} U$
$T \xrightarrow{\Phi(T,R)\le 0.1} R_0$
Generate L_{l+1}
$C^+(TI) = T$ $C^+(TU) = T$ $C^+(TR_0) = T$
$C^+(IU) = R$ $C^+(IR_0) = R$ $C^+(UR_0) = R$
$l = 2$
Iteration 2
IU, TI, TU, TR_0 ⇒output
are appr_keys
$IU \xrightarrow{\Phi(IU,T)\le 0.2} T$
$IU \xrightarrow{\Phi(IU,R_0)\le 0.3} R_0$
$C^+(TIU) = \emptyset$ $C^+(TIR_0) = T$
$C^+(TUR_0) = T$ $C^+(IUR_0) = \{U, I\}$
$l = 3$
Iteration 3
Search DAFDs
$IR_0 \xrightarrow{\Phi(IU,R_0)\le 0.3} U$
$UR_0 \xrightarrow{\Phi(IU,R_0)\le 0.3} I$
Update $C^+(X)$
$C^+(TIU) = \emptyset$ $C^+(TIR_0) = T$
$C^+(TUR_0) = T$ $C^+(IUR_0) = \emptyset$
$l = 4$
$L_l \neq \emptyset$?
No
Yes
Non-trivial and minimal DAFDs
$T \xrightarrow{\Phi(T,I)\le 0.1} I$
$T \xrightarrow{\Phi(T,U)\le 0.1} U$
$T \xrightarrow{\Phi(T,R)\le 0.1} R_0$
$IU \xrightarrow{\Phi(IU,T)\le 0.2} T$
$IU \xrightarrow{\Phi(IU,R_0)\le 0.3} R_0$
$IR_0 \xrightarrow{\Phi(IU,R_0)\le 0.3} U$
$UR_0 \xrightarrow{\Phi(IU,R_0)\le 0.3} I$

Figure 1: Demonstration of the DAFD discovering process

Theorem 4.4. *Given the mining result Σ obtained from DAFDDiscover, each DAFD in Σ is a non-trivial and minimal DAFD.*

Proof. (**Non-trivial**). From DAFDDiscover, each DAFD in Σ comes from two procedures, `Compute_dependencies` and `Prune`. As the output of `Compute_ dependencies`, DAFDs are shaped like $X\backslash\{A\} \xrightarrow{\Psi(X\backslash\{A\},A)\le\sum_{A_i\in X} a_i} A$, and are decidedly non-trivial. The form of DAFD output by `Prune` is like $X \xrightarrow{\Psi(X,A)\le\sum_{A_i\in X\cup A} a_i} A$, where $A \notin X$ (line 5 in Algorithm 3), so it is non-trivial. In summary, every DAFD in Σ is a non-trivial DAFD. (**Minimal**). Assuming that there exists a DAFD of the form $X \xrightarrow{\Psi(X,A)\le\alpha(X,A)} A \in \Sigma$, where $\exists Z \subset X, Z \xrightarrow{\Psi(Z,A)\le\alpha(Z,A)} A$ holds. However, according to `Generate_next_level`, which generates L_l in ascending order of set cardinality, $Z \xrightarrow{\Psi\le\alpha} A$ will be judged to be true before $X \xrightarrow{\Psi\le\alpha} A$, leading to the pruning of $X \xrightarrow{\Psi\le\alpha} A$. This means that $X \xrightarrow{\Psi\le\alpha} A$ cannot appear in Σ, contradicting the fundamental assumption in Section 3.2.1. Thus, there does not exist any subset Z of X such that $Z \xrightarrow{\Psi\le\alpha} A$ holds. In other words, all attributes in $X \xrightarrow{\Psi\le\alpha} A$ are necessary, and there are no redundant attributes in any DAFD in Σ. Thus, all DAFDs in Σ are minimal DAFDs. □

After ensuring that each DFAD in the result set satisfies the requirements in Problem 1, we prove the correctness and completeness of the mining results of the proposed DAFDDiscover.

Theorem 4.5. *All DAFDs in Σ are valid, and any DAFD that actually holds on $\mathcal{I}_D$ can be derived from DAFDs in Σ.*

Proof. Each DAFD in Σ comes from two procedures, `Prune` and `Compute_dependencies`. Since the LHS of a DAFD from `Prune` is a super key, the LHS value of each tuple is different, so the DAFD will not be violated. DAFDs output by `Prune` must be valid. The DAFDs from `Compute_Dependencies` are valid by the DAFD definition, so they must be valid. In summary, all the DAFDs in Σ are valid.

Since DAFDDiscover essentially traverts the whole search space by three ways: definition determination, key attribute property and armstrong axiom derivation determination, the mining process visits every valid DAFD and make a decision. Among them, the DAFDs through definition decision and key attribute property decision is output to Σ. The DAFDs decided by armstrong axiom system is pruned directly. Since pruning other than key pruning depends on armstrong axiom system, the pruned DAFDs can be obtained from the DAFDs in Σ by armstrong axiom system. In summary, any DAFD that holds for $\mathcal{I}_D$ can be derived from the DAFDs in Σ. □

We can see that our DAFD mining results ensure correctness and comprehensiveness. Synthesizing all theorems of Section 3 and Section 4, Theorem 4.6 holds.

Theorem 4.6. *The corresponding same-shaped FD of every DAFD in Σ is a nontrivial minimal FD on $\mathcal{I}_D$ that actually holds.*

It is a significant conclusion for direct computation on low-quality data that the mining results of the proposed DAFDDiscover can accurately reflect the real FDs present in data. Consequently, both DAFDDiscover and DAFDDiscover+ are effective in mining the dependencies hidden within dirty data.

4.4 Analysis of the time and space complexity

Considering the time and space performance of both DAFDDiscover and DAFDDiscover+, we present the conclusion in Theorem 4.7.

Theorem 4.7. *The upper bound on the time complexity and the space complexity of DAFDDiscover and DAFDDiscover+ are equivalent to those of the Tane algorithm.*

Proof. (1) **Time complexity**. For data $\mathcal{I}_D$ with n tuples and m attributes, DAFDDiscover has a time complexity of $O(n)$ for each dependency validation test. Compared to Tane, DAFDDiscover's single valid test only requires an additional calculation of the threshold $\sum_{A_i\in X} a_i$, which costs $O(m+n)$. On large datasets where the number of tuples significantly exceeds the number of attributes, it has $O(m+n) = O(n)$. Thus, the time complexity of a single valid test is the same for both DAFDDiscover and Tane.

Further, DAFDDiscover adopts a more aggressive pruning strategy based on Armstrong's axiom system compared to Tane's pruning strategy in AFD mining, DAFDDiscover traverses the entire search space faster, resulting in an overall time complexity that does not exceed that of TANE. As for DAFDISCOVER+, the time cost of a single DAFD candidate's threshold is reduced from $O(m)$ to $O(1)$, and it employs an even more aggressive pruning strategy. Its overall time complexity does not exceed that of DAFDISCOVER. Therefore, DAFDISCOVER+ also has an overall time complexity that does not exceed that of TANE.

Specifically, the partition component of TANE costs $O(n2^m)$, the pruning costs $O(n^{2.5}2^m)$, and both `Compute_dependencies` and `Generation_ next_ level` cost $O(m2^m)$, TANE spends $O(2^m mn + 2^m m^2 + 2^m m^3) = O(2^m(mn+m^{2.5}))$ for AFD mining. DAFDISCOVER costs $O(2^m(m+n)m + 2^m m^2 + 2^m m^3) = O(2^m(mn + m^{2.5}))$, and DAFDISCOVER+ also costs $O(2^m(mn + m^{2.5}))$ in time.

(2) **Space complexity**. DAFDISCOVER requires additional storage for the upper limit values of the proportion of dirty data for m attributes. Given that the space complexity of TANE is $O(\frac{(m+n)2^m}{\sqrt{m}})$, DAFDISCOVER costs $O(\frac{(m+n)2^m}{\sqrt{m}} + m)$ in space. Since $\sqrt{m}2^m > m$, it has $O(\frac{(m+n)2^m}{\sqrt{m}} + m) = O(\frac{(m+n)2^m}{\sqrt{m}})$, indicating that DAFDISCOVER has the same upper bound on space complexity as TANE.

DAFDISCOVER+ requires more space to store the decision thresholds for each attribute set. As the number of attribute sets in the search space is of size $O(2^m)$, the space complexity for storing the attribute set decision thresholds in the DAFDISCOVER+ is $O(2^m)$. Considering that DAFDISCOVER+ adopts a similar breadth-first traversal strategy to TANE, the space complexity for storing attribute set decision thresholds can be reduced to $O(\frac{2^m}{\sqrt{m}})$[15]. Thus, the overall space complexity of DAFDISCOVER+ is $O(\frac{(m+n)2^m}{\sqrt{m}} + \frac{2^m}{\sqrt{m}}) = O(\frac{(m+n)2^m}{\sqrt{m}})$. This means that DAFDISCOVER+ also has the same upper bound on space complexity as TANE. □

4.5 Robustness Analysis

Next, we conduct an in-depth analysis of the robustness of the algorithm, which is a crucial aspect of the DAFD mining problem.

4.5.1 Overview of the Robustness. DAFDISCOVER utilizes the mapping relationship between DAFD and FD to mine data dependencies within dirty data, leveraging the input metric of *the upper limit of the proportion of erroneous dat*a from each attribute's corresponding data source. However, in real-world scenarios, it is often impossible to definitively assert that the proportion of erroneous data from a specific data source. Instead, it is typically ensured that the proportion of erroneous data is less than a certain value with a certain probability. Given the uncertainties inherent in dirty data, the mining results of DAFDISCOVER cannot *absolutely* guarantee accurate reflections of the underlying dependencies within the dirty data. Nevertheless, we emphasize that if DAFDISCOVER can produce reliable mining results for upper limits on errors that hold with sufficiently high probability, this would indicate the robustness of DAFDISCOVER w.r.t the upper limit of the proportion of dirty values.

We propose a novel metric to describe the reliability of DAFD, namely the probability of DAFD validity (DAFD-*prob*), in order to evaluate the robustness of the mining results. By considering both support and the probability of validity, we demonstrate in this section that for input upper limits that hold with sufficiently high probability, DAFDISCOVER achieves reliable mining results.

4.5.2 DAFD-prob: The probability of validity for the mining results. Recall Example 1.1, it underscores that for the same dataset, different attributes with varying data sources may lead to differing reliability assessments. This necessitates that the evaluation of DAFD reliability should incorporate considerations of data source characteristics, as relying solely on support and confidence metrics based on the dataset alone may not achieve reliability assessment.

Given knowledge of the data source properties, one can consider calculating the probability of an FD corresponding to a DAFD being valid (according to Theorem 4.6). A higher probability of validity indicates a greater likelihood that the isomorphic FD corresponding to DAFD is valid, justifying its reliability. Conversely, a lower probability suggests that the isomorphic FD is unlikely to hold, casting doubt on the reliability of DAFD. As data source conditions can be factored into the calculation of the probability of validity, using it as a metric for assessing DAFD reliability is a reasonable choice. Therefore, we propose using the probability of FD validity corresponding to DAFD as a metric for evaluating DAFD reliability, denoted as DAFD-*prob*. The calculation of DAFD-*prob* necessitates scenario-specific data analysis. Below, we present a method for calculating DAFD-*prob* under the assumptions made in this paper and provide a theoretical lower bound expression for it.

DAFD-*prob*. According to Theorem 3.2, if the event "*there exists an upper bound $k = a_i n$ for the number of erroneous tuples under this attribute*" holds, then for any DAFD candidate, it is certain that the event "*there is an upper bound $\tau = \sum_{A_i \in X \cup Y} a_i n$ for the number of erroneous data tuples*" (denoted as event E_A) will occur. In this case, when a DAFD is valid on the original data, the isomorphic FD on the underlying true data of the dataset is also valid. Therefore, for an FD candidate ψ_{FD} and its corresponding isomorphic DAFD candidate φ, by applying the total probability formula, it has: *DAFD-prob* $= \Pr(\models \psi_{FD}) \geq \Pr(\models \psi_{FD} | E_A \textit{and} \models \varphi) \cdot \Pr(\models \varphi | E_A) \cdot \Pr(E_A)$. According to Theorem 4.6, $\Pr(\models \psi_{FD} | E_A \textit{and} \models \varphi) = 1$. When the algorithm determines that $\mathcal{I}_D \models \varphi$, then $\Pr(\models \varphi | E_A) = 1$. Consequently, it follows that $\Pr(\models \psi_{FD}) \geq \Pr(E_A)$.

Next, we calculate $\Pr(E_A)$. For the dependency $X \rightarrow Y$, let p denote the probability that a tuple in the projection of the dataset onto $X \cup Y$ contains dirty data. Using the total probability formula, for any attribute sets $A, B \subseteq X \cup Y$, we have: $\Pr(B \text{ clean}) = \Pr(A \text{ clean}) \Pr(B \text{ clean}|A \text{ clean}) + \Pr(A is \text{ dirty}) \Pr(B \text{ clean}|A is \text{ dirty})$. Based on assumptions in Section 3.2.1, it holds that $\Pr(B \text{ clean}) \geq \Pr(B \text{ clean}|A is \text{ dirty})$, which implies $\Pr(B \text{ clean}) \leq \Pr(B \text{ clean}|A \text{ clean})$. Furthermore, it can be shown that:

$$p = 1 - \prod_{i=1}^{m} \Pr(A_i \text{ clean}|A_1 \ldots A_{i-1} \text{ clean}) \leq 1 - \prod_{i=1}^{m} P(A_i \text{ clean}) = p_0.$$

In fact, each cell can only be correct or incorrect, and each tuple can only contain erroneous data or not. Therefore, the number of tuples with erroneous data follows a binomial distribution. Specifically, for the number of erroneous tuples τ_0 in the low-quality dataset, we have $\tau_0 \sim B(n, p)$, where n is the number of tuples in the dataset. As $n \rightarrow +\infty$, by the Central Limit Theorem, the binomial distribution can be approximated by a normal distribution, i.e., $\tau_0 \sim N(np, np(1-p))$.

When $n \to +\infty$, let S denote the probability of the event "*the upper bound for the number of tuples with erroneous data is* τ". It has

$$\Pr(E_A) = S = \sum_{i=1}^{\tau} C_n^{\tau} p^{\tau} (1-p)^{\tau} \approx \int_{-\infty}^{\tau} f(x)dx,$$

where $f(x)$ is the probability density function of $N(np, np(1-p))$. Let $f_0(x)$ be the probability density function of $N(np_0, np_0(1-p_0))$. When $\tau \geq np$ and $p \leq p_0 < 0.5$, it holds that:

$$S = \int_{-\infty}^{\tau} f(x)dx = \frac{1}{2} + \int_{np}^{\tau} f(x)dx \tag{1}$$

$$= \frac{1}{2} + \int_{np_0}^{np_0 + \frac{(\tau - np)}{\sqrt{np(1-p)}}\sqrt{np_0(1-p_0)}} f_0(x)dx \tag{2}$$

$$\geq \frac{1}{2} + \int_{np_0}^{\tau} f_0(x)dx \tag{3}$$

Based on the maximum likelihood method, the existence of np tuples with erroneous data in the dataset is considered most probable. Therefore, $\tau < np$ indicates an unreasonable setting for τ. In fact, τ should be generated by a learning algorithm, rather than exposed directly to users, to avoid issues with an unreasonable setting of τ. Furthermore, since the probability of error for each attribute is sufficiently small, it is evident that $p_0 < 0.5$.

In summary, the following conclusion holds:

$$\text{DAFD-}prob = \Pr(\models \psi_{FD}) \geq \Pr(E_A) \geq \frac{1}{2} + \int_{np_0}^{\tau} f_0(x)dx.$$

When τ is known, it is possible to calculate a lower bound for the probability that ψ_{FD} is valid, i.e., a lower bound for DAFD-*prob*.

4.5.3 Robustness Evaluation of DAFDiscover. The above equation determines the lower bound of the reliability of the mining results, which is dictated by the nature of the data source, given a fixed number of tuples violating the dependency on the original dataset. However, intuitively, the characteristics of the data itself should also be considered as part of the reliability assessment. That is, even when dealing with dirty datasets, people tend to believe that a dependency with fewer violations is more reliable. Therefore, the reliability of the mining results should be conducted from the perspectives of *support* and *probability* of validity.

Consider a DAFD φ in the result set Σ with support denoted as *sup*. The upper limit of the proportion of erroneous data for attribute A_i corresponding to the data source is determined based on the 3σ principle. From the view of support, based on the definition of DAFD, it is easy to see that the support of φ satisfies:

$$sup \geq 1 - \frac{\sum_{A_i \in X \cup Y}(a_i n + 3\sqrt{n a_i (1-a_i)})}{n}$$

where $X \cup Y \subseteq Attr(\mathcal{R})$, and thus $|X \cup Y| \leq m$. Therefore, when m is a relatively small constant and n is sufficiently large, based on assumptions in Section 3.2.1, $\sum_{A_i \in X \cup Y} a_i$ should be sufficiently small, resulting in a sufficiently large *sup* for φ. Table 3 illustrates the relationship between m, a_i, and *sup*. It can be seen that under the conditions of a relatively small m, sufficiently small a_i, and a sufficiently large n, φ has a sufficiently large support. Hence, from the perspective of support, the result Σ is reliable.

Note that the values of *sup* in Table 3 actually represent the lower bounds of the support for φ, and the actual support is often greater than the values shown in Table 3. Additionally, in real datasets, there may be order-of-magnitude differences in a_i for different attributes, and Table 3 has been simplified for ease of estimation. It reports that when the order of magnitude of $\sum a_i$ is not greater than 10^{-3}, the mining results tend to have relatively ideal support.

Table 3: Relationship between m, a_i, and *sup* (assuming an order of magnitude of n as 10^4)

m	10	10	10	10^2	10^2	10^2	10^4	10^4	10^4
a_i:	10^{-2}	10^{-3}	10^{-4}	10^{-3}	10^{-4}	10^{-5}	10^{-4}	10^{-5}	10^{-6}
sup:	0.87	0.98	0.966	0.8	0.96	0.989	0.6	0.89	0.969

Table 4: Summary of datasets

Dataset	#*Tuples*	#*Attributes*	#*FDs*
Abalone[26]	4177	9	137
Chess[4]	28056	7	1
Breast-cancer[34]	699	11	46
Forestfires[11]	517	13	442
Air-quality[31]	9358	15	1765
Bike-sharing[13]	731	16	519
Diabetic[25]	2278	10	55
Raisin[9]	900	8	43
Bitcoinheist[1]	2916697	10	-
Caulkins[10]	1685	12	227
Hughes[10]	400	8	3

From the perspective of DAFD-*prob*, it has:

$$\text{DAFD-}prob = \frac{1}{2} + \int_{np_0}^{np_0 + \frac{(\tau - np)}{\sqrt{np(1-p)}}\sqrt{np_0(1-p_0)}} f_0(x)dx \tag{4}$$

$$\geq \frac{1}{2} + \int_{np_0}^{np_0 + 3\sqrt{np_0(1-p_0)}} f_0(x)dx \geq 0.997 \tag{5}$$

Here, $\tau = \sum_{A_i \in X \cup Y}(a_i n + 3\sqrt{n a_i (1-a_i)})$. The corresponding proof is provided in our extended paper in Github.

Proof. To prove (4) and (5), we only need to prove $\frac{t-np}{\sqrt{np(1-p)}}$ is no less than 3.

In fact, we have

$$\frac{t-np}{\sqrt{np(1-p)}} = \frac{\sum_{i \in X}(a_i n + 3\sqrt{n a_i (1-a_i)}) - np}{\sqrt{np(1-p)}}$$
$$\geq \frac{n\sum_{i \in X} a_i + 3\sum_{i \in X}\sqrt{n a_i (1-a_i)} - n(1 - \prod_{i \in X}(1-a_i))}{\sqrt{np(1-p)}}$$

Since there exists $1+x_1+x_2+\cdots+x_n \leq (1+x_1)(1+x_2)\ldots(1+x_n)$ for all $x_i > -1 (i = 1, 2, \ldots, n)$, we have

$$\frac{t-np}{\sqrt{np(1-p)}} \geq \frac{n\sum_{i \in X} a_i + 3\sum_{i \in X}\sqrt{n a_i (1-a_i)} - n(1 - \prod_{i \in X}(1-a_i))}{\sqrt{np(1-p)}}$$
$$\geq \frac{n\sum_{i \in X} a_i + 3\sum_{i \in X}\sqrt{n a_i (1-a_i)} - n + n(1 - \sum_{i \in X} a_i)}{\sqrt{np(1-p)}}$$
$$= 3 \cdot \frac{\sum_{i \in X}\sqrt{a_i(1-a_i)}}{\sqrt{p(1-p)}}$$

Since the probability of making mistakes for each attribute is low enough, we assume that $p_0 < 0.5$. In this situation, we have

$$\begin{aligned}\frac{t-np}{\sqrt{np(1-p)}} &\geq 3\cdot\frac{\sum_{i\in X}\sqrt{a_i(1-a_i)}}{\sqrt{p(1-p)}}\\ &\geq 3\cdot\frac{\sum_{i\in X}\sqrt{a_i(1-a_i)}}{\sqrt{p_0(1-p_0)}}\\ &= 3\cdot\frac{\sum_{i\in X}\sqrt{a_i(1-a_i)}}{\sqrt{[1-\prod_{i\in X}(1-a_i)]\prod_{i\in X}(1-a_i)}}\end{aligned}$$

Since there exists $(\sum_{i\in X}\sqrt{a_i})^2 \geq \sum_{i\in X} a_i$, we have

$$\begin{aligned}\frac{t-np}{\sqrt{np(1-p)}} &\geq 3\cdot\frac{\sum_{i\in X}\sqrt{a_i(1-a_i)}}{\sqrt{[1-\prod_{i\in X}(1-a_i)]\prod_{i\in X}(1-a_i)}}\\ &\geq 3\cdot\sqrt{\frac{\sum_{i\in X}a_i(1-a_i)}{[1-\prod_{i\in X}(1-a_i)]\prod_{i\in X}(1-a_i)}}\\ &\geq 3\cdot\sqrt{\frac{\sum_{i\in X}a_i(1-a_i)}{(\sum_{i\in X}a_i)\prod_{i\in X}(1-a_i)}}\\ &\geq 3\cdot\sqrt{\frac{(1-max_{i\in X}(a_i))\sum_{i\in X}a_i}{(\sum_{i\in X}a_i)\prod_{i\in X}(1-a_i)}}\\ &= 3\cdot\sqrt{\frac{(1-max_{i\in X}(a_i))}{\prod_{i\in X}(1-a_i)}}\\ &\geq 3\end{aligned}$$

So, we can say that $\frac{t-np}{\sqrt{np(1-p)}}$ is no less than 3, which can prove (4) and (5). □

Accordingly, for an input upper bound that holds with sufficiently high probability, DAFDiscover obtains mining results with a sufficiently high probability of validity. Thus, we confirm the reliability of the DAFDiscover mining results from the perspective of the probability of validity, which further shows that the DAFD discovery solution proposed in this paper exhibits robustness.

5 EXPERIMENTAL EVALUATION

We introduce the experimental setup in Section 5.1 and subsequently present experimental results in the following subsections.

5.1 Experimental setting.

Experimental dataset. We employed 11 real-world datasets, most of which are commonly used and considered classics in the data dependency mining field, as summarized in Table 4.

Comparison Methods. We implement all algorithms introduced in this paper, referring to the entire cleaning process as RFDiscover. Besides the proposed method, we implement the following baseline discovery methods.

• Tane, a classical AFD mining algorithm based on the schema-driven strategy [15]. • Pyro: A SOTA AFD mining method that traverses the entire search space through ascend and trick-down phases, combined with stripped partitions to mine all AFDs in dataset[18].

• HyFD: A full-scale FD mining algorithm that utilizes a hybrid strategy[28].

• FDx: An FD mining method based on structure learning for noisy data. It achieves dependency mining on noisy datasets through a structure learning approach[38].

• MiDD: [23] introduces the mutual information score metric $\frac{I(X,Y)}{I(Y,Y)}$ to evaluate dependencies on noisy data, but does not provide a mining algorithm. We implement a brute-force mining algorithm based on the mutual information metric to discover all dependencies where $\frac{I(X,Y)}{I(Y,Y)}$ exceeds a certain threshold.

• SoFD: A naive mining algorithm specifically designed for soft functional dependencies[17].

Metrics. We evaluate the performance of algorithms using three types of metrics. (1) Quality of the mining results is assessed by precision $P = \frac{\#\varphi(\text{valid\&discovered})}{\#\varphi(\text{discovered})}$, recall $R = \frac{\#\varphi(\text{valid\&discovered})}{\#\varphi(\text{valid})}$, $F1 = \frac{2\cdot P\cdot R}{P+R}$, and the number (Num) of dependencies discovered. (2) Scalability is measured using two indicators: algorithm execution *time* and maximum *memory* usage. (3) The ability of DAFD to reflect actual FDs in the presence of potentially unreliable error-prone data is evaluated using DAFD-*prob* and support metrics.

Implementation. Considering the current lack of standard datasets designed for evaluating dependencies directly on dirty data, we enhance the aforementioned real datasets primarily through amplification and noise injection. For data enhancement, we employ data duplication (i.e., data×128) to achieve amplification and adopt the noise injection method of adding "*" from Bart[3] to introduce errors. Noise injection involves injecting faulty data into each attribute according to a certain upper limit or probability (*noi%*), simulating errors from the corresponding data sources.

It is important to note that for the calculation of metric (1), the results of HyFD and Pyro use their own mining outcomes on clean datasets as the benchmark for comparison. FDx does not have a suitable comparison baseline, thus P, R, and $F1$ are not calculated for it. For the remaining algorithms, the FD mining results of Tane algorithm on clean datasets serve as the baseline.

5.2 Overall performance evaluation

We validate the overall performance of the mining results in Table 5, demonstrate that both DAFDiscover and the DAFDiscover+ yield identical mining outcomes. Under the fundamental assumptions outlined in the paper, these algorithms achieve perfect P and R (both at 1) in mining dependency relationships from low-quality data. Furthermore, it reveals that Tane performs well in terms of P and R compared to other SOTA methods. This confirms the reliability and validity of our choice to use Tane as the foundation for constructing the DAFD mining approach.

While MiDD exhibits high precision, its recall is relatively low. SoFD and HyFD display moderate performance in both P and R. These observations highlight the limitations of existing SOTA methods that rely on fixed parameters to control the tolerance for violating tuples. Even though adjusting these parameters can potentially improve aP and R, the tuning process itself introduces additional operational costs and reduces the algorithms' usability. The results further emphasize the necessity of incorporating dynamic thresholds into AFDs.

Table 5: Overall performance comparison

Methods	*Breast-cancer*					*Chess*					*Forestfires*					*Abalone*					*Raisin*				
	P	*R*	*F*1	*Num*	**Time**	*P*	*R*	*F*1	*Num*	**Time**	*P*	*R*	*F*1	*Num*	**Time**	*P*	*R*	*F*1	*Num*	**Time**	*P*	*R*	*F*1	*Num*	**Time**
DAFDiscover	1	1	1	89	9.84	1	1	1	18	11.88	1	1	1	140	31.69	1	1	1	130	14.93	1	1	1	49	1.27
DAFDiscover+	1	1	1	89	9.49	1	1	1	18	10.82	1	1	1	140	29.95	1	1	1	130	14.94	1	1	1	49	1.32
Tane	1	0.88	0.94	79	10.52	1	1	1	18	11.17	1	1	1	140	24.59	1	1	1	130	13.75	0.81	0.55	0.67	32	1.76
Pyro	0.01	0.01	0.01	104	1.28	1	1	1	49	0.69	0.03	0.04	0.03	102	5.25	0.01	0.03	0.02	80	2.11	0.02	0.07	0.03	61	0.44
HyFD	0.81	0.51	0.63	49	2.95	1	0.5	0.67	1	6.99	0.79	0.64	0.71	88	1.81	0.60	0.57	0.59	111	11.46	0.8	0.43	0.56	20	0.96
MiDD	0.92	0.71	0.83	71	38.96	1	0.66	0.8	12	28.62	0.93	0.82	0.88	124	114.52	1	0.89	0.94	116	22.31	0.8	0.40	0.54	25	3.83
SoFD	0.63	0.67	0.65	95	15.41	1	0.66	0.8	12	12.89	0.47	0.60	0.53	179	50.66	0.47	0.46	0.47	127	20.76	0.8	0.40	0.54	25	2.06
FDx	-	-	-	10	0.84	-	-	-	4	5.77	-	-	-	12	6.38	-	-	-	8	3.41	-	-	-	7	0.77

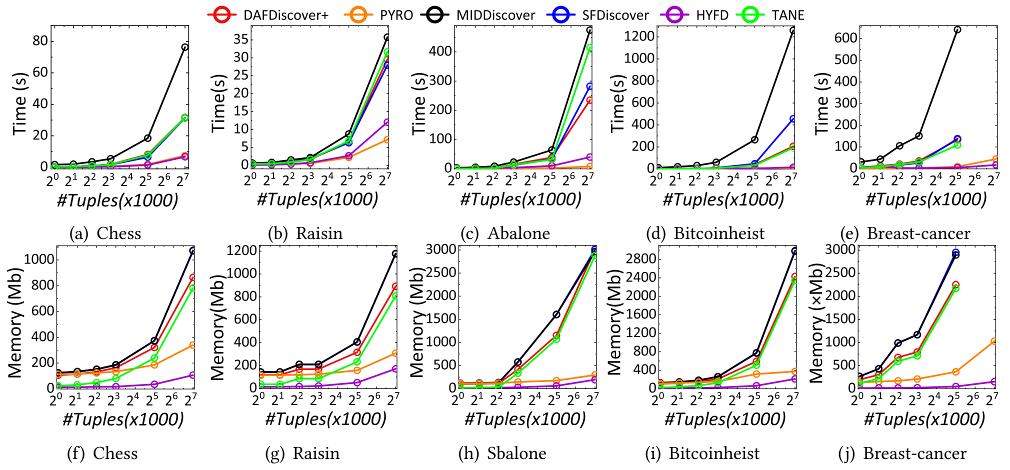


Figure 2: Scalability evaluation with varying tuples (under data amplification of datasets)

Table 6: Robustness verification (with noise injection in attributes)

Dataset	#col	#row	DAFD instances	DAFD-*prob*	sup
Breast-cancer	11	699	*Scn,Mitoses⟶NoNu*	0.9986	0.9871
noi%:			*Scn,NoNu⟶Mitoses*	0.9986	0.9971
NoNu:0.005			*Scn,Mitoses⟶Class*	0.9988	0.9986
Class: 0.001			*Scn,ClTh⟶ NoNu*	0.9986	0.9987
			Scn,ClTh⟶ Class	0.9988	0.9986
Caulkins	12	1685	*L'n,Drug⟶ DrCd*	0.9999	1
noi%:			*Drug,AbPu⟶ DrCd*	0.9987	0.9994
QinG:0.01			*Drug,QinG⟶ DrCd*	0.9986	0.9982
HiPu: 0.005			*Drug,HiPu⟶ DrCd*	0.9986	1
AbPu:0.001			*QinG,Pure⟶ AbPu*	0.9999	0.9958
Hughes	8	401	*hr,grp⟶ confirm*	0.9986	0.9776
noi%: test:0.02			*dtime,grp⟶ dstatus*	0.9986	0.9875
dstatus: 0.005			*h1,hr,grp⟶ dstatus*	0.9986	0.9825
confirm: 0.005			*dtime,grp⟶ confirm*	0.9986	0.9850
result: 0.01			*h1,grp,test⟶ dstatus*	0.9986	0.9776

Table 7: Ablation evaluation of the key pruning strategy

	Abalone			*Raisin*			*Bitcoinheist*		
	P	*R*	*Num*	*P*	*R*	*Num*	*P*	*R*	*Num*
DAFDiscover	1	1	246	1	1	44	1	1	69
DAFDiscover+	1	1	246	1	1	44	1	1	69
nTane	0.91	0.75	202	0.89	0.89	44	0.97	0.93	66
nTane+	0.96	0.87	225	1	0.98	43	1	0.93	64
Tane-base	-	-	246	-	-	44	-	-	69

5.3 Robustness verification of DAFDiscover

We simulate a noisy data environment by introducing noise with a fixed probability to several attributes of the dataset. The experimental results are summarized in Table 6, where we presents part of mined DAFD results. The results show that the DAFD-*prob* values of the DAFDs discovered by DAFDiscover are not less than 0.997, which aligns with the theoretical analysis presented in Section 4.5.2. Additionally, the support values of the mining results are not less than 0.95. These findings indicate that the mining results exhibit good performance in both DAFD-*prob* and support metrics, demonstrating the high reliability of the proposed DAFDiscover when dealing with low-quality data.

5.4 Scalability evaluation with varying tuples

We evaluate the scalability performance on five datasets with data amplification, as the results depicted in Figure 2. Regarding time performance, DAFDiscover+ exhibits similar trends and comparable time costs across all datasets. The time costs of DAFDiscover+ on Chess increase linearly with the number of tuples, aligning with our expectations. However, on Abalone and Breast-cancer, while the time costs initially increase linearly for up to 2^5K tuples, there is a sharp rise in execution time for Abalone at that point, and a deceleration in growth rate for Breast-cancer beyond 4000 tuples.

In terms of space performance, the trends of DAFDiscover+ change on Abalone and Breast-cancer. Similar patterns appear in the schema-driven strategies of Tane, MiDD, and SoFD, we hypothesize that these variations are due to alterations in the pruning process during traversal induced by data amplification and memory limitations when handling larger tuple counts. Tane exhibits the lowest time and space costs due to its aggressive pruning strategy and lack of need to store additional information for attribute sets at each node. Conversely, MiDD and SoFD incur higher time and space overhead due to their more conservative pruning approaches.

HyFD even shows a decrease in runtime with increasing tuples, attributed to fewer iterations in the mining process resulting from dependency variations. The absolute time and space costs of HyFD

Table 8: A case study comparing the effectiveness of dependency mining on dirty data

Dataset	Dependencies	DAFDiscover	HyFD	Tane	Pyro	SoFD	FDx	MiDD
Air-quality	temperature → temperature_suitability (valid)	√		√	√			√
	relative_humidity → humidity_comfort_level (invalid)	√	√			√	√	
Bike-sharing	month → quarter (valid)	√		√				√
	weekday → workingday (invalid)	√	√			√	√	
Diabetic	RIDAGEYR → age_group (valid)	√		√				
	LBXIN → DIQ010 (invalid)	√	√				√	

and Pyro are also lower. While DAFDiscover+ has higher consumption than HyFD and Pyro due to the added computational cost of calculating dynamic tolerance thresholds, overall, DAFD more accurately captures dependencies underlying dirty data compared to FD and AFD.

5.5 Ablation study on pruning step

We conduct an ablation study on the key pruning step, comparing three strategies: (1) nTane, which substitutes the key pruning approach in DAFDiscover with that of Tane; (2) nTane+, which modifies the pruning strategy in DAFDiscover+ to immediately remove nodes in the subsequent level that contain all attributes of a node deemed as an approximate superkey during pruning. This mirrors Tane's pruning principle but extends it to approximate superkeys. We used the mining results of Tane-base, derived from DAFDiscover by eliminating key pruning, as the baseline for comparison. As shown in Table 7, while the Tane's key pruning sometimes leads to slightly reduced execution time, it can result in erroneous pruning (as analyzed in Section 4.2.2). Conversely, the key pruning strategies employed in DAFDiscover and DAFDiscover+ yield identical mining outcomes as the unpruned algorithm, highlighting the necessity of our proposed key pruning approach in designing accurate DAFD mining algorithms.

5.6 Case study

We compare the effectiveness of DAFD and other FD forms in discovering dependencies on dirty data. We evaluate various FD forms based on their ability to correctly discover known semantically dependent relationships and accurately exclude invalid dependencies.

Results in Table 8 show DAFDiscover consistently and accurately identifies valid dependencies while avoiding false positives in noisy datasets. Other algorithms, however, exhibit varying degrees of misjudgment or error. HyFD fails to recognize temperature → temperature_suitability due to dirty data present in the attribute, highlighting the lack of robustness of the FD rule form against dirty data. While Tane and Pyro miss the invalidity of relative_humidity → humidity_comfort_level, because the constant ϵ used in AFD is significantly higher than the upper limit of erroneous data for humiditycomfortlevel. This results in AFD falsely considering the dependency valid despite the high proportion of violations.

SoFD and MiDD incorrectly handle these dependencies, reflecting the limitations of using fixed thresholds to statically determine tolerance for low-quality data in expressing dependencies. Furthermore, FDx's failure to correctly identify the temperature → temperaturesuitability relationships indicates that the structure learning strategy alone cannot fully address these limitations inherent in static threshold-based approaches.

On bike-sharing, HyFD misses Mnth → quarter due to its inflexibility in tolerating noise. Tane falsely identifies weekday → workingday as a dependency while overlooking Mnth → quarter. This is because, despite weekday → workingday not being a strict functional dependency, the low frequency of holidays in the dataset causes the number of tuples violating this dependency to be less than Tane's threshold, leading to incorrect inclusion. Pyro exhibits the same issue. Similar to bike-sharing, Both SoFD and MiDD not only fail to discover the correct dependencies but also misjudge the validity of LBXIN → DIQ010 in the diabetic dataset. Like FD, which does not relax its satisfaction criteria, and AFD mining algorithms with thresholds that are not well-suited to real datasets, they cannot accurately judge both relationships in low-quality data as effectively as DAFD.

These results demonstrate that the propose DAFD can effectively adapt to dirty data with significant variations in the upper limit of erroneous data proportions across different attributes. In fact, even when existing RFDs like AFD and SFD can adapt to mining tasks on certain datasets, they require parameter determination and adjustment, incurring additional time and labor costs. Conversely, DAFD obviates the need for pre-tuning, saving on both human and time resources while maintaining a robust mining performance across a wider range of scenarios. MI fails to exclude the invalid dependency weekday → workingday due to inappropriate static parameter settings, highlighting the limitations of using fixed thresholds in tolerating inferior data when expressing dependencies.

6 CONCLUSIONS

This paper introduces DAFD, a novel AFD form that dynamically tolerates dirty data by incorporating data source information. We analyze its properties and establish its mapping relationship with traditional FDs. We propose DAFDiscover, an algorithm for mining DAFDs, and investigate the characteristics of its mining results. Experimental results demonstrate the superiority of our approach in dependency mining from dirty data. Future research directions include (1) continuously optimizing the time complexity of DAFDiscover and (2) extending the concept of dynamic AFD to other more expressive quality constraints, such as denial constraints.

REFERENCES

[1] 2020. BitcoinHeistRansomwareAddressDataset. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5BG8V.

[2] Ziawasch Abedjan, Patrick Schulze, and Felix Naumann. 2014. DFD: Efficient Functional Dependency Discovery. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, CIKM 2014, Shanghai, China, November 3-7, 2014*, Jianzhong Li, Xiaoyang Sean Wang, Minos N. Garofalakis, Ian Soboroff, Torsten Suel, and Min Wang (Eds.). ACM, 949–958. https://doi.org/10.1145/2661829.2661884

[3] Patricia C. Arocena, Boris Glavic, Giansalvatore Mecca, Renée J. Miller, Paolo Papotti, and Donatello Santoro. 2015. Messing Up with BART: Error Generation for Evaluating Data-Cleaning Algorithms. *Proc. VLDB Endow.* 9, 2 (2015), 36–47. https://doi.org/10.14778/2850578.2850579

[4] Michael Bain and Arthur Hoff. 1994. Chess (King-Rook vs. King). UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C57W2S.

[5] Tobias Bleifuß, Susanne Bülow, Johannes Frohnhofen, Julian Risch, Georg Wiese, Sebastian Kruse, Thorsten Papenbrock, and Felix Naumann. 2016. Approximate Discovery of Functional Dependencies for Large Datasets. In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management, CIKM 2016, Indianapolis, IN, USA, October 24-28, 2016*, Snehasis Mukhopadhyay, ChengXiang Zhai, Elisa Bertino, Fabio Crestani, Javed Mostafa, Jie Tang, Luo Si, Xiaofang Zhou, Yi Chang, Yunyao Li, and Parikshit Sondhi (Eds.). ACM,

1803–1812. https://doi.org/10.1145/2983323.2983781

[6] Loredana Caruccio, Vincenzo Deufemia, Felix Naumann, and Giuseppe Polese. 2021. Discovering Relaxed Functional Dependencies Based on Multi-Attribute Dominance. *IEEE Trans. Knowl. Data Eng.* 33, 9 (2021), 3212–3228. https://doi.org/10.1109/TKDE.2020.2967722

[7] Loredana Caruccio, Vincenzo Deufemia, and Giuseppe Polese. 2016. Relaxed Functional Dependencies - A Survey of Approaches. *IEEE Trans. Knowl. Data Eng.* 28, 1 (2016), 147–165.

[8] Fei Chiang and Renée J. Miller. 2008. Discovering data quality rules. *Proc. VLDB Endow.* 1, 1 (2008), 1166–1177. http://www.vldb.org/pvldb/vol1/1453980.pdf

[9] Koklu Murat Cinar, Ilkay and Sakir Tasdemir. 2023. Raisin. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5660T.

[10] codocedo. 2018. tane. https://github.com/codocedo/tane.

[11] Paulo Cortez and Anbal Morais. 2008. Forest Fires. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5D88D.

[12] Wenfei Fan and Floris Geerts. 2012. *Foundations of Data Quality Management.* Morgan & Claypool Publishers.

[13] Hadi Fanaee-T. 2013. Bike Sharing Dataset. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5W894.

[14] Peter A. Flach and Iztok Savnik. 1999. Database Dependency Discovery: A Machine Learning Approach. *AI Commun.* 12, 3 (1999), 139–160. http://content.iospress.com/articles/ai-communications/aic182

[15] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. 1999. TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies. *Comput. J.* 42, 2 (1999), 100–111. https://doi.org/10.1093/COMJNL/42.2.100

[16] Ihab F. Ilyas and Xu Chu. 2019. *Data Cleaning.* ACM.

[17] Ihab F. Ilyas, Volker Markl, Peter J. Haas, Paul Brown, and Ashraf Aboulnaga. 2004. CORDS: Automatic Discovery of Correlations and Soft Functional Dependencies. In *SIGMOD*, Gerhard Weikum, Arnd Christian König, and Stefan Deßloch (Eds.). ACM, 647–658.

[18] Sebastian Kruse and Felix Naumann. 2018. Efficient Discovery of Approximate Dependencies. *Proc. VLDB Endow.* 11, 7 (2018), 759–772. https://doi.org/10.14778/3192965.3192968

[19] Qiongqiong Lin, Yunfan Gu, Jingyan Sai, Jinfei Liu, Kui Ren, Li Xiong, Tianzhen Wang, Yanbei Pang, Sheng Wang, and Feifei Li. 2023. EulerFD: An Efficient Double-Cycle Approximation of Functional Dependencies. In *39th IEEE International Conference on Data Engineering, ICDE 2023, Anaheim, CA, USA, April 3-7, 2023*. IEEE, 2878–2891. https://doi.org/10.1109/ICDE55515.2023.00220

[20] Jixue Liu, Jiuyong Li, Chengfei Liu, and Yongfeng Chen. 2012. Discover Dependencies from Data - A Review. *IEEE Trans. Knowl. Data Eng.* 24, 2 (2012), 251–264. https://doi.org/10.1109/TKDE.2010.197

[21] Stéphane Lopes, Jean-Marc Petit, and Lotfi Lakhal. 2000. Efficient Discovery of Functional Dependencies and Armstrong Relations. In *Advances in Database Technology - EDBT 2000, 7th International Conference on Extending Database Technology, Konstanz, Germany, March 27-31, 2000, Proceedings (Lecture Notes in Computer Science)*, Carlo Zaniolo, Peter C. Lockemann, Marc H. Scholl, and Torsten Grust (Eds.), Vol. 1777. Springer, 350–364. https://doi.org/10.1007/3-540-46439-5_24

[22] Panagiotis Mandros, Mario Boley, and Jilles Vreeken. 2017. Discovering Reliable Approximate Functional Dependencies. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*. ACM, 355–363. https://doi.org/10.1145/3097983.3098062

[23] Panagiotis Mandros, Mario Boley, and Jilles Vreeken. 2018. Discovering Reliable Dependencies from Data: Hardness and Improved Algorithms. In *2018 IEEE International Conference on Data Mining (ICDM)*. 317–326. https://doi.org/10.1109/ICDM.2018.00047

[24] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. 1997. Discovery of Frequent Episodes in Event Sequences. *Data Min. Knowl. Discov.* 1, 3 (1997), 259–289. https://doi.org/10.1023/A:1009748302351

[25] NA NA. 2023. National Health and Nutrition Health Survey 2013-2014 (NHANES) Age Prediction Subset. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5BS66.

[26] Sellers Tracy Talbot Simon Cawthorn Andrew Nash, Warwick and Wes Ford. 1995. Abalone. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C55C7W.

[27] Noël Novelli and Rosine Cicchetti. 2001. FUN: An Efficient Algorithm for Mining Functional and Embedded Dependencies. In *Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings (Lecture Notes in Computer Science)*, Jan Van den Bussche and Victor Vianu (Eds.), Vol. 1973. Springer, 189–203. https://doi.org/10.1007/3-540-44503-X_13

[28] Thorsten Papenbrock and Felix Naumann. 2016. A Hybrid Approach to Functional Dependency Discovery. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.). ACM, 821–833. https://doi.org/10.1145/2882903.2915203

[29] Frédéric Pennerath, Panagiotis Mandros, and Jilles Vreeken. 2020. Discovering Approximate Functional Dependencies using Smoothed Mutual Information. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash (Eds.). ACM, 1254–1264. https://doi.org/10.1145/3394486.3403178

[30] Shaoxu Song, Fei Gao, Ruihong Huang, and Chaokun Wang. 2022. Data Dependencies Extended for Variety and Veracity: A Family Tree. *IEEE Trans. Knowl. Data Eng.* 34, 10 (2022), 4717–4736. https://doi.org/10.1109/TKDE.2020.3046443

[31] Saverio Vito. 2016. Air Quality. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C59K5F.

[32] Daisy Zhe Wang, Xin Luna Dong, Anish Das Sarma, Michael J. Franklin, and Alon Y. Halevy. 2009. Functional Dependency Generation and Applications in Pay-As-You-Go Data Integration Systems. In *12th International Workshop on the Web and Databases, WebDB 2009, Providence, Rhode Island, USA, June 28, 2009*. http://webdb09.cse.buffalo.edu/papers/Paper18/webdb09.pdf

[33] Ziheng Wei and Sebastian Link. 2019. Discovery and Ranking of Functional Dependencies. In *35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, April 8-11, 2019*. IEEE, 1526–1537. https://doi.org/10.1109/ICDE.2019.00137

[34] WIlliam Wolberg. 1992. Breast Cancer Wisconsin (Original). UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5HP4Z.

[35] Catharine M. Wyss, Chris Giannella, and Edward L. Robertson. 2001. FastFDs: A Heuristic-Driven, Depth-First Algorithm for Mining Functional Dependencies from Relation Instances. In *Data Warehousing and Knowledge Discovery*, Yahiko Kambayashi, Werner Winiwarter, and Masatoshi Arikawa (Eds.). 101–110. https://doi.org/10.1007/3-540-44801-2_11

[36] Renjie Xiao, Yong'an Yuan, Zijing Tan, Shuai Ma, and Wei Wang. 2022. Dynamic Functional Dependency Discovery with Dynamic Hitting Set Enumeration. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*. IEEE, 286–298. https://doi.org/10.1109/ICDE53745.2022.00026

[37] Hong Yao, Howard J. Hamilton, and Cory J. Butz. 2002. FD_Mine: Discovering Functional Dependencies in a Database Using Equivalences. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9-12 December 2002, Maebashi City, Japan*. IEEE Computer Society, 729–732. https://doi.org/10.1109/ICDM.2002.1184040

[38] Yunjia Zhang, Zhihan Guo, and Theodoros Rekatsinas. 2020. A Statistical Perspective on Discovering Functional Dependencies in Noisy Data. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 861–876. https://doi.org/10.1145/3318464.3389749