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 [9, 17, 32]. Functional dependencies (FDs) [13, 16], 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, 32]).

Conventionally, exact FDs are utilized to describe the relationship models between data. With the rapid accumulation of data in information systems, issues such as noise, vacancies, and redundancies are prevalent in real-world datasets. R4D1 To enhance the applicability of FDs in the context of big data, relaxed functional dependencies (RFDs) have been introduced, allowing for some degree of relaxation (e.g., the extent of dependency satisfaction) [7]. A RFD can be concisely represented as $X_{\Phi_1} \xrightarrow{\Psi \leq \epsilon} Y_{\Phi_2}$ [7]. 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. Compared to exact FDs, RFDs exhibit greater adaptability to real-world big data in many scenarios [8, 32]. R4W1 R4D1 Specifically, RFDs offer the following key advantages: (1) greater flexibility in defining relationships between attributes, (2) stronger fault tolerance in data quality management due to less stringent accuracy requirements, (3) obvious time savings by reducing the need for rigorous data verification and correction, and (4) the potential to uncover associations and patterns that might be overlooked under traditional FDs due to their allowance for more data variation.

The dependency $X \xrightarrow{\Psi \leq \epsilon} Y$, which allows relaxation in satisfaction degree, is commonly referred to as approximate functional dependency (AFD). R4D2 It provides a mechanism that can enhance tolerance for *noise* in large datasets. However, we have observed that existing AFD mining algorithms typically rely on a fixed hyperparameter ϵ [5, 20], for determining and mining dependencies from datasets. R1W1 R1D1 The limitation of such a static relaxation coefficient in AFD becomes apparent when dealing with complex, low-quality big data. It is reflected in the following aspects. 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. R1D6 A motivated example illustrates this issue.

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

Example 1.1. Table 1 presents monitoring records for an electrical appliance, recording 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. 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 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: $I_0 \rightarrow I_1$. Additionally, I_0 is determined by both U_0 and R_0, leading to FD_2: $U_0R_0 \rightarrow I_0$. When attempting to mine these 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 AFD, 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, conducting reliable dependency mining on dirty data using dynamic AFD is not a straightforward task. The difficulties arise from two main sources. Firstly, DAFD inherits the same challenges as AFD mining, such as a vast solution space and difficulties in pruning, posing challenges to the time performance of the mining algorithm. Secondly, due to its support for more flexible and dynamic relaxation thresholds, DAFD mining faces unique challenges, primarily including the following aspects. R1W1

▹ **Additional threshold calculation**: As an extension of AFD, DAFD requires threshold computation for each candidate during mining, incurring an extra $O(m^2 \cdot 2^m)$ time cost for a dataset with m attributes. Optimizing these threshold computations to minimize this overhead is a challenge in designing DAFD mining algorithms.

▹ **Evaluation of the mining results on dirty 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 introduce dynamic functional dependencies to solve the problem of direct dependency mining on dirty data. Contributions include:

(1) We introduce DAFD, an extended AFD variant, which is better suited to represent the quality rules underlying data containing dirty values in attributes. We establish the bijective relationship between DAFD on dirty data and FD on clean data, and prove the reflexivity, augmentation, and transitivity properties of DAFD. R1D5

(2) We propose the mining algorithm named DAFDiscover. It accurately captures dependencies on dirty data with the proposed DAFD form, and has the same time and space complexity with the classical AFD mining algorithm Tane. DAFDiscover+ is introduced to optimize 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, denoted as *DAFD-prob*, to validate the mining results' validity probability on dirty data. We prove that, under basic assumptions, this probability is at least $\frac{1}{2}+\int_{np_0}^{n\cdot\sum_{A_i\in X\cup Y}\alpha_i} f_0(x)dx$, where n is the total number of tuples and $X \cup Y$ covers all dependency elements.

(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 are key integrity constraints in databases. Broadly, the automation of FD mining algorithms includes schema-driven, instance-driven, and hybrid approaches [17]. 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[16], Fun[29], and FD_Mine[40], 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 include FDep[15], FastFD, and Dep-Miner[23]. Hybrid strategies aim to combine the strengths of both schema-driven and instance-driven methods. These algorithms maintain high efficiency in confirming or eliminating FD candidates, e.g., HyFD [30] and DhyFD[37].

Mining complete FDs often has exponential time complexity, limiting real-world use. Approximate and dynamic FD mining are explored as potential solutions, sacrificing minimality accuracy for efficiency, e.g., via methods like Aid-FD [5] and EulerFD[21].

Approximate FD discovery. Tane[16] also proposes AFD discovery solution, which calculates the minimum number of tuples to be deleted using stripped partitions. It employs a breadth-first strategy to traverse the search space effectively. Pyro[20] 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[18], utilizes a sampling strategy for mining soft functional dependencies (SoFD). Meanwhile, a method for computing partial FD (PFD) is introduced in [35]. Furthermore, [6] presents Domino, which focuses on mining RFDs with a more lenient comparison approach.

R1W1 R1D1 AFD exhibits stronger mining capabilities in complex and dynamic big data environments due to their relaxed satisfaction criteria. However, in scenarios where the data inherently contains errors, the use of a static and fixed relaxation threshold demonstrates limitations. Consequently, to accommodate the characteristics of dirty data, researchers have also excluded non-RFD mining approaches. [24] 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 and discover top-k dependencies based on the metric. However, since the number of dependencies is often not known, the top-k strategy fail to provide a completeness set of dependencies. cannot find all dependencies. FDx [41] is proposed to target noisy data, by framing the FD mining problem as a structure learning task. Note that FDx only find a dependency for each attribute, it cannot ensure to find all dependencies on the dirty data.

R1W1 R1D1 AFD semantics often struggle to adapt to real-world data containing erroneous information. Additionally, the few algorithms capable of directly mining on dirty data still lack solid theoretical guarantees. 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.

3 OVERVIEW OF DAFD

3.1 Preliminaries

$Attr(\mathcal{R}) = (A_1, ..., A_m)$ represents the set of attributes in a database relation $\mathcal{R}$. $\mathcal{I}$ is an instance of $\mathcal{R}$ 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 **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. One **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 that quantifies the amount of tuples violating or satisfying φ, and ϵ is a threshold indicating its upper bound.

R1W2 R1D2 Aiming to better support the discovery of dependency patterns in low-quality data containing dirty values, we introduce a dynamic relaxation (e.g., $\alpha(X, Y)$) instead of a fixed one from the perspective of satisfaction degree. Definition 3.1 formalizes our DAFD form.

Definition 3.1. **Dynamic Approximate Functional Dependencies (DAFD)**. Given $\mathcal{I}_D$ as a low-quality data instance of $\mathcal{R}$ containing dirty values, $X \xrightarrow{\Psi(X,Y) \leq \alpha(X,Y)} Y$ represents 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 $\forall y \in Y$, $\Psi(X, y) \leq \alpha(X, y)$.

R1W2 R1D2 Here, the degree of dependency satisfaction is measured by $\Psi(X, Y) = \frac{\min\left\{|\mathcal{I}_v| \,|\, \mathcal{I}_v \subseteq \mathcal{I}_D \text{ and } X \rightarrow Y \text{ holds in } \mathcal{I}_D \setminus \mathcal{I}_v\right\}}{|\mathcal{I}_D|}$. It denotes the ratio of the minimum number of tuples that need to be removed for the dependency $X \rightarrow Y$ to hold in a subset of tuples $\mathcal{I}_v$ from $\mathcal{I}_D$, to the total number of tuples in the dataset (i.e., $|\mathcal{I}_D|$).

And a_i is the upper bound on the proportion of erroneous data for the corresponding attribute in data $\mathcal{I}_D$, which reflects the inherent nature of the data source itself. The error threshold $\alpha(X, Y) = \sum_{A_i \in X \cup Y} a_i$ denotes the upper bound of the threshold for the allowable proportion of dirty data, which means that at most a proportion of $\sum_{A_i \in X \cup Y} a_i$ of the total tuples in the attribute set $X \cup Y$ corresponding to $X \rightarrow Y$ can contain erroneous data.

Note that the definition of DAFD based on $|Y|$ could be reflected to the set of FDs behind the dirty dataset one by one, which is beneficial to the accurate discovery for the FDs behind the dirty dataset. We prove this characterization in Theorem 3.3. R1D2

Example 3.2. For the dependency φ: $I_0 \rightarrow I_1$ in Example 1.1, given the maximum error limits for these two attributes as $a_{I_0} = 0.1$ and $a_{I_1} = 0.2$, the upper limit for tolerable dirty data proportion is $\alpha(I_0, I_1) = a_{I_0} + a_{I_1} = 0.3$. We calculate the proportion of records that do not satisfy φ as $\Psi(I_0, I_1) = \frac{3}{10} = 0.3$ from Table 1. Since $\Psi(I_0, I_1) \leq \alpha(I_0, I_1)$, DAFD $I_0 \xrightarrow{\Psi(I_0,I_1) < \alpha(I_0,I_1)} I_1$ holds on this dataset. In contrast, the general AFD takes the form $I_0 \xrightarrow{\Psi(I_0,I_1) < \epsilon} I_1$, where ϵ is a predefined threshold. If improperly set, such as $\epsilon = 0.1$, we may easily overlook a dependency like $I_0 \rightarrow I_1$. R1W2 R1D3 R1D6

3.2 The nature of DAFD

3.2.1 The fundamental assumption regarding datasets containing dirty values. Regarding our research subject, which involves a dataset $\mathcal{I}_D$ containing dirty values, we first provide a formal description and make the following assumptions based on real-world scenarios [19, 41]: (*a*) The error probability of each attribute's data source is relatively low, thus implying the existence of a small upper bound on the proportion of erroneous data for each attribute. (*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. R1W2

We believe that the assumptions regarding dirty datasets possess applicability and rationality in real-world scenarios. Regarding assumption (*a*), information systems have evolved over the years, and in many contexts (such as the Internet of Vehicles, patient vital sign monitoring, and air quality monitoring systems [19, 33]), data acquisition technology has achieved relatively stable performance. Although erroneous records cannot be avoided due to environmental changes or transmission interruptions, the proportion of incorrect data in information systems remains relatively small. As for assumption (*b*), we adopt the underlying logic of approximate dependency mining, which trusts the patterns or regularities reflected by the majority of the data while treating minor outliers or errors as exceptions. Hence, when considering datasets containing erroneous values, if a FD candidate does not hold in this low-quality dataset, and the proportion of tuples violating this FD significantly exceeds the proportion of tuples containing erroneous attribute values, it indicates that the number of tuples violating the FD has far surpassed what can be explained solely by data errors. In other words, such a high proportion of violations is more likely due to R1W2 R1W3 R1D4-1 R4D3 R1D4-2

the invalidity of the FD candidate itself, rather than merely being caused by data errors.

3.2.2 DAFD vs. FD. For the DAFD in Definition 3.1, when setting the allowable threshold $\alpha(X, Y) = \sum_{A_i \in X \cup Y} a_i$, we obtain the relationship between DAFD and its corresponding FD below.

R1W2 R1D5 THEOREM 3.3. *Let $\mathcal{I}$ be a completely clean dataset on the relation schema $\mathcal{R}$, and $\mathcal{I}_D$ be the low-quality version of $\mathcal{I}$ on $\mathcal{R}$ which satisfies the assumptions above, a DAFD holds on $\mathcal{I}_D$ if and only if the corresponding isomorphic FD holds on $\mathcal{I}$.*

R1D6 R4W1 R4D4 *Here, 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, as shown in Figure 1.*

PROOF. Given a DAFD φ, (i) when $|RHS(\varphi)| = 1$, if φ does not hold on $\mathcal{I}_D$, even if all tuples with erroneous data in $\mathcal{I}_D$ violate φ, there must still be some tuples with entirely correct data that violate φ (otherwise, φ 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 φ 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 φ holds on $\mathcal{I}_D$, then $\forall y \in RHS(\varphi)$, φ 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 $\mathcal{I}$. Thus, the FD holds on $\mathcal{I}$. If φ does not hold on $\mathcal{I}_D$, then there exists a $y \in RHS(\varphi)$ such that φ formed by the LHS and y does not hold. By (i), $\exists y \in RHS(\varphi)$ 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 the clean data $\mathcal{I}$ if and only if the corresponding DAFD holds on the dirty data $\mathcal{I}_D$. □

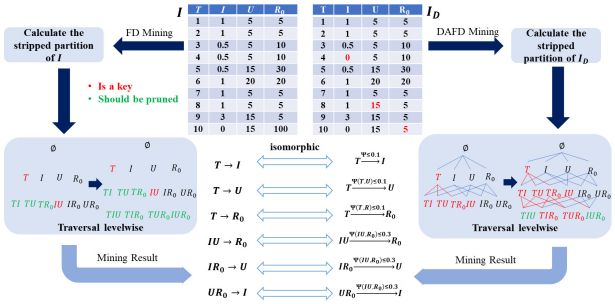

Figure 1: The isomorphism of FDs on $\mathcal{I}$ and DAFDs on $\mathcal{I}_D$.

3.2.3 DAFD reasoning. Theorem 3.4 introduces the inference system of DAFD, which is crucial for the mining of DAFD from data.

R1W2 R1D5 THEOREM 3.4. *Let $\mathcal{I}_D$ be the dirty version of the clean data $\mathcal{I}$, DAFD satisfies the Armstrong axiom system on $\mathcal{I}_D$. That is, DAFD satisfies the law of reflexive, augmentation, and transitive on $\mathcal{I}_D$.*

PROOF. **Reflexive**. Based on the property of FD [16], $X \rightarrow X$ is always true. Furthermore, according to Theorem 3.3, $X \xrightarrow{\Psi(X,X) \leq \alpha(X,X)}$ X holds true consistently on $\mathcal{I}_D$. **Augmentation**. If $X \xrightarrow{\Psi(X,Y) \leq \alpha(X,Y)}$ Y is valid on $\mathcal{I}_D$, then according to Theorem 3.3, $X \rightarrow Y$ is valid on $\mathcal{I}$. Further, based on the properties of FD, $XA \rightarrow YA$ ($\forall A \in Attr(\mathcal{R})$) on $\mathcal{I}$. Thus, $XA \xrightarrow{\Psi(XA,YA) \leq \alpha(XA,YA)}$ YA is also valid on $\mathcal{I}_D$. **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 3.3, $X \rightarrow Y$ 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 valid $\mathcal{I}_D$. □

3.2.4 A comparative analysis of the properties of DAFD and other FD variations. Table 2 summarizes FD and several of its variations, alongside their representative mining algorithms. In comparison, R1W1 DAFD offers the following advantages: (1) *The dependency semantic expression*: While AFD, PFD, and SoFD support a fixed level of error R2W1 tolerance in terms of satisfiability, the proposed DAFD supports R2D1 a more flexible threshold that takes into account the probability of attribute value errors. This allows for a better capture of data dependency patterns within dirty datasets, thereby reflecting the underlying FDs more accurately. (2) *Reasoning properties*: Compared to the existing AFD, DAFD possesses clearer and more explicit reasoning properties. (3) *Theoretical guarantees* of the mining algorithm: The mining approach for DAFD, specifically DAFDiscover introduced in Section 4, not only achieves competitive time and space complexity but also theoretically ensures the validity, minimality, and completeness of the mining results. This provides a significant advantage for the efficient mining of DAFD.

4 DISCOVERING DAFDS

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 algorithms' time and space complexity, as well as the 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 $\forall \varphi \in \Sigma_{\text{DAFD}}$, φ hold true in $\mathcal{I}_D$. (ii) One DAFD $\varphi: X \xrightarrow{\Psi(X,Y) \leq \alpha(X,Y)} Y$ is* R2D2 ***minimal** iff $\forall X' \subset X$, $X' \xrightarrow{\Psi(X',Y) \leq \alpha(X',Y)} Y$ is not valid, and (iii) one DAFD $\varphi: X \xrightarrow{\Psi(X,Y) \leq \alpha(X,Y)} Y$ is **non-trivial** iff $Y \notin X$.*

According to Theorem 3.3, there exists a one-to-one correspondence between DAFD on$\mathcal{I}_D$ and FD on the corresponding clean $\mathcal{I}$. Therefore, Theorem 4.1 holds true for the DAFD mining problem.

THEOREM 4.1. *Let Σ_{DAFD} denote the mining result for DAFDs on a dirty dataset $\mathcal{I}_D$. Σ_{FD} represents the set of FDs that are isomorphic to the DAFDs in Σ_{DAFD}, then Σ_{FD} corresponds to the mining result for FDs on the associated clean dataset $\mathcal{I}$ from which $\mathcal{I}_D$ originates.*

Complexity discussion. The time complexity of FD mining is bounded by $O(n^2(\frac{m}{2})^2 2^m)$, where n is the number of tuples and m is the number of attributes [22]. DAFD mining introduces additional computational overhead due to dynamic threshold calculations. It traverse the search space analogously to FD mining, with each dependency requiring $O(n)$ time for evaluation. Its total time complexity,

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[16]	$O(2^m(n+m^{2.5}))$	$O(\frac{(n+m)2^m}{\sqrt{m}})$	All FDs
							FastFD[39]	-	$O(\lvert Difference\_set\rvert m)$	All FDs
							HYFD[30]	$O(mn^2+m^22^m)$	-	All FDs
						Approximate Discovery	EulerFD[21]	almost linearly with row expansion	-	All FDs
							AID-FD[5]	-	-	All FDs
	extent of satisfaction	AFD	$X \xrightarrow{\frac{\lvert I_D\rvert-\max\{\lvert t\rvert t\subseteq I_D \text{ and } X\to Y \text{ holds in } t\}}{\lvert I_D\rvert}\le\epsilon} Y$	Fixed	$X\to Y \Rightarrow XA\to Y$	TANE[16]		$O(2^m(mn+m^{2.5}))$	$O(\frac{(n+m)2^m}{\sqrt{m}})$	All AFDs
			$X \xrightarrow{\frac{\lvert\{(t_1,t_2)\in I_D^2\rvert t_1[X]=t_2[X]\wedge t_1[Y]\neq t_2[Y]\}\rvert}{\lvert I_D\rvert^2-\lvert I_D\rvert}\le\epsilon} Y$	Fixed	$X\to Y \Rightarrow XA\to Y$	PYRO[20]		-	-	All AFDs
		PFD[35]	$X \xrightarrow{\frac{\sum_{x\in dom(X)}P(X\to Y,Vx)}{\lvert dom(X)\rvert}\ge 1-\epsilon} Y$	Fixed	unknown	extension of TANE[35]		-	-	-
		SoFD[18]	$X \xrightarrow{\frac{\lvert dom(X)\rvert_{I_D}}{\lvert dom(X,Y)\rvert_{I_D}}\ge 1-\epsilon} Y$	Fixed	$X\to Y \Rightarrow XA\to Y$	CORDS[18]		-	-	SoFDs with single attribute in LHS
		DAFD	$X \xrightarrow{\frac{\min\{\lvert t\rvert t\subseteq I_D \text{ and } X\to Y \text{ holds in } I_D\backslash t\}}{\lvert I_D\rvert}\le\alpha(X,Y)} Y$	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[24]		-	-	Top-k FD discovery
						SMI[31]		-	-	
Structure learning			$X\to Y$, with the set of attributes X in $\mathcal{R}$ that correspond to non-zero entries in autoregression matrix[41]	based on the process of learning	-	FDX[41]		quadratic complexity with respect to the number of columns	-	One FD for each RHS attribute

including dependency evaluation and threshold computations, is $O(n^3(\frac{m}{2})^2 2^m + T(m,n))$, where $T(m,n)$ represents the cumulative time for threshold computations.

DAFD discovery, like FD and AFD, faces challenges in effectively pruning the $O(m\cdot 2^m)$ search space. However, DAFD's relaxed satisfaction criteria, allowing minor data violations (i.e., dirty values), complicate simultaneous evaluation and pruning of multiple dependency candidates. Each DAFD candidate requires threshold computation, adding $T(m,n)$ to mining time. A brute-force approach yields $T(m,n) = O(m^2 2^m)$, emphasizing the need for effective pruning strategies to reduce $T(m,n)$.

4.2 DAFDiscover algorithm

4.2.1 The holistic design of DAFDiscover. Insights from Section 3.2.4 reveal that DAFDs are semantically akin to AFDs, differing only in that DAFDs use a dependency function in place of AFDs' ϵ. This suggests that state-of-the-art AFD mining algorithms can be suitably adapted for DAFD mining. Specifically, replacing ϵ with the function $\alpha(X,Y) = \sum_{A_i\in X\cup Y} a_i$ within any AFD mining algorithm could facilitate DAFD discovery. Notably, unlike AFDs, DAFDs adhere to the Armstrong axiom system, enabling more aggressive pruning strategies for efficient search space reduction. This offers potential improvements in the efficiency and efficacy of DAFD mining. 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 DAFDiscover. In a review of the prevalent AFD mining algorithms, Tane [16] stands out as a schema-driven approach that employs a breadth-first strategy to traverse the 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 $X \xrightarrow{\Psi(X,A)\le\epsilon} A$ holds, then $\forall B \in Attr(\mathcal{R})$, $XB \xrightarrow{\Psi(XB,A)\le\epsilon} A$ holds. (ii) If $X \xrightarrow{\Psi(X,A)\le\epsilon} A$ is a valid FD, then $\forall B \in Attr(\mathcal{R})$, $XB \xrightarrow{\Psi(XB,A)\le\epsilon} A$ 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 $\forall B \in Attr(\mathcal{R})$, $X \xrightarrow{\Psi(X,B)\le\epsilon} B$ 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 $C^+(X) = \emptyset$.

Hence, our initial approach is to adapt Tane for DAFD mining. Since DAFDs conform to the Armstrong axiom system, we can employ TANE's original pruning techniques for DAFD 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}. For example, let $AB \xrightarrow{*} C$ and $BC \xrightarrow{*} A$ be DAFDs on $\mathcal{I}_D$, where BC is a key but AB is not. According to Tane's key pruning strategy, since BC is a key, when Prune visits BC, it will output $BC \to A$ and remove BC from L_l, preventing the generation of node ABC. This results in the DAFD $AB \to C$ not being computed. Additionally, because AB is not a key, the algorithm will not output $AB \to C$ during the key check for AB. Consequently, $AB \to C$ is lost during the mining process. R2W4

Specifically, all DAFD of the form $X \xrightarrow{\Psi\le\alpha} A$ and $(X\cup\{A\}) - \{B\} \xrightarrow{\Psi\le\alpha} B$ 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 $X \xrightarrow{\Psi\le\alpha} A$ are valid and have been discovered in L_l. It does not guarantee the validity of DAFD of the form $(X\cup\{A\})-\{B\} \xrightarrow{\Psi\le\alpha} B$, nor does it ensure that the valid ones can be discovered in L_l on attribute sets other than the removed $X\cup A$.

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 $X \xrightarrow{\Psi\le\alpha} A$ 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

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, Σ_{DAFD}

$L_0 \leftarrow \{\emptyset\}, C^+(\emptyset) \leftarrow Attr(\mathcal{R}), L_1 \leftarrow \big\{\{A\} | A \in Attr(\mathcal{R})\big\}$;
$l \leftarrow 1$;
foreach $X \in L_1$ **do**
  $C^+(X) \leftarrow \bigcap_{A \in X} C^+(X \setminus \{A\})$;
while $L_1 \neq \emptyset$ **do**
  Compute_dependencies(L_l);
  Prune(L_l);
  $L_{l+1} \leftarrow$ Generate_next_level(L_l), $l \leftarrow l + 1$;

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.

From the above, we propose a DAFD mining algorithm with a novel key pruning strategy specifically designed for DAFD mining on dirty data. 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 (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 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.

Algorithm 2: Compute_dependencies(L_l)

Input : a set of attribute sets L_l with l attributes
Output: the remaining DAFDs on L_l

foreach $X \in L_l$ **do**
  foreach $A \in X \cap C^+(X)$ **do**
    if $e(X \setminus \{A\} \rightarrow A) \leq \sum_{A_i \in X} a_i)$ **then**
      return $X \setminus \{A\} \xrightarrow{\Psi(X\setminus\{A\},A) \leq \sum_{A_i \in X} a_i} A$;
      remove A from $C^+(X)$;
      remove all B in $R \setminus X$ from $C^+(X)$;

The Prune function, 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 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

foreach $X \in L_l$ **do**
  if $C^+(X) = \emptyset$ **then**
    delete X from L_l;
  if X *is an (super) key* **then**
    foreach $A \in C^+(X) \setminus X$ **do**
      if $A \in \bigcap_{B \in X} C^+(X \cup \{A\} \setminus \{B\})$ **then**
        return $X \rightarrow A$;
    Record X as a key;

The Generate_next_level(L_l) function, 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[26].

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}

$L_{l+1} \leftarrow \emptyset$;
foreach $K \in$ Prefix_Blocks(L_l) **do**
  foreach $\{Y, Z\} \subseteq K, (Y \neq Z)$ **do**
    $X \leftarrow Y \cup Z, C^+(X) \leftarrow \bigcap_{A \in X} C^+(X \setminus \{A\})$;
    if X_is_valid(L_l, X) **then**
      $L_{l+1} \leftarrow L_{l+1} \cup \{X\}$;
return L_{l+1};

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}.

4.2.3 Optimization of DAFDiscover. We emphasize that there are still possibilities for further improvements to DAFDiscover. We identify 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 allowance 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 $X \setminus \{A\} \rightarrow A$

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 ∈ X do
2      check ← 0;
3      for aset ∈ L do
4          if aset = X \ {x} then
5              check ← 1;
6              if aset is key then
7                  remove A from C+(X);
8                  remove all B in Attr(R) \ X from C+(X);
9              Break;
10         if check = 0 then
11             return False;
12 if C+(X) = ∅ then
13     return False;
14 return True;
```

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 $X \setminus \{A\} \rightarrow A$ for each X, we can optimize by computing the corresponding 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 $O(m2^m)$, which is implemented in `Compute_dependencies`, and requires only a minor modification to the first four lines in Algorithm 2 as follows.

```
1': foreach X ∈ L_i do
2':     ε_X = Σ_{A_i ∈ X} a_i;
3':     foreach A ∈ X ∩ C+(X) do
4':         if e(X \ {A} → A) ≤ ε_X then
5':             return X \ {A} --Ψ(X\{A},A) ≤ ε_X--> A;
```

That is, when calculating the DAFDs for any set X in L_l, we first compute 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 `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 \ {A} → A) ≤ ε_X then
```

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

```
5':  if X_is_valid(L_l, X) then
6':      ε_X ← ε_Y + ε_{X\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 executes pruning solely for exact superkeys. If X is an *approximate superkey*, meaning it can Transform into a superkey by eliminating a small fraction of tuple, i.e., $\leq \sum_{A_i \in X} a_i$, its violating tuples as a DAFD candidate will not exceed this proportion. Hence, for any $Y^* \in Attr(\mathcal{R})$ added to X, this proportion remains valid. This guarantees X can form a valid DAFD as the LHS. So, pruning approximate superkeys is viable, allowing for a more aggressive pruning strategy than 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[16] 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 derive DAFDiscover+, an improved version of DAFDiscover. The general framework of DAFDiscover+ resembles the original DAFDiscover, with the only adjustment being the initialization step in Algorithm 1 line 1 as follows:

```
1': L_0 ← {(∅, 0)}, C+(∅) ← Attr(R), L_1 ← {(A_i, a_i) | A_i ∈ Attr(R)};
```

Example 4.2. Fig. 2 illustrates the mining process for the data in Example 1.1. 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 $X \setminus \{A\} \rightarrow A$ for each attribute set in L_l (i.e., attribute sets with l attributes at level l) using `Compute_dependencies`, e.g., $IR_0 \rightarrow U$ in the attribute set IUR_0. 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 (e.g., IUR_0 in L_3 shown in green in Fig. 2) and identifies and marks key attribute sets (e.g., T in L_1 and IU in L_2 in red). The corresponding DAFDs for the key attribute sets are output. R1D6

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 the 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 $\mathcal{I}_D$, the mining results of DAFDiscover+ are identical to those of the DAFDiscover.*

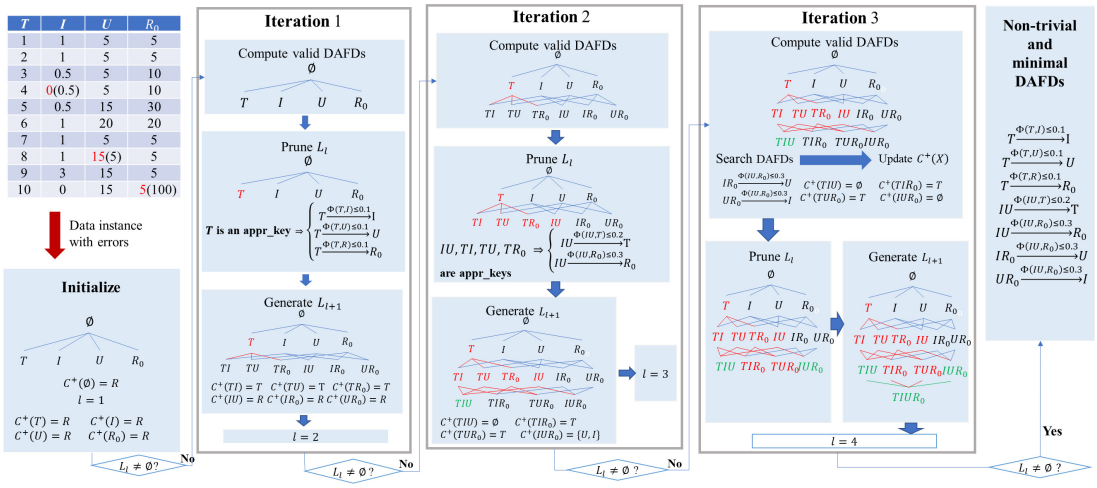

Figure 2: Demonstration of the DAFD discovering process from dirty data

Proof. Denoting the output sets of DAFDiscover and DAFDiscover+ as Σ and Σ^+ respectively, we observe that DAFDiscover+ incorporates two enhancements: refined threshold computation and the extension of key pruning to include approximate keys. The refined threshold process eliminates redundant calculations without altering the threshold values, ensuring Σ and Σ^+.

In the second enhancement, approximate key pruning, which is a generalized form of key pruning, ensures that all dependencies identified through key pruning are included in the output. If an attribute set X qualifies as an approximate key, removing a maximum of $\sum_{A_i \in X} a_i$ tuples converts it into a key. This implies that for any DAFD candidate $X \rightarrow A$, $\Psi(X, A) \leq \sum_{A_i \in X} a_i \leq \sum_{A_i \in X \cup A} a_i$, satisfying the DAFD criteria. Therefore, all pruned candidates adhere to the DAFD definition and would be captured by DAFDiscover+. The optimization does not cause differences between Σ and Σ^+. R2W2

In summary, the optimizations in DAFDiscover+ maintain the equivalence of Σ and Σ^+, ensuring consistent mining results between both algorithms. □

Since the mining results of these two algorithms are identical, we analyze the properties of the mining results obtained by DAFDiscover with Theorem 4.4-4.6.

Theorem 4.4. *Given the mining result Σ obtained from DAFDiscover, each DAFD in Σ is a **non-trivial** and **minimal** DAFD.* R1W1 R1D1

Proof. (**Non-trivial**). From DAFDiscover, each DAFD in Σ comes from two procedures, `Compute_dependencies` and `Prune`. As the output of `Compute_ dependencies`, DAFDs are shaped like $X\backslash\{A\} \xrightarrow{\Psi(X\backslash\{A\},A) \leq \sum_{A_i \in X} a_i} A$, and are decidedly non-trivial. The form of DAFD output by `Prune` is like $X \xrightarrow{\Psi(X,A) \leq \sum_{A_i \in X \cup A} a_i} A$, 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 $X \xrightarrow{\Psi(X,A) \leq \alpha(X,A)} A \in \Sigma$, where $\exists Z \subset X, Z \xrightarrow{\Psi(Z,A) \leq \alpha(Z,A)} A$ holds. However, according to `Generate_next_level`, which generates L_l in ascending order of set cardinality, $Z \rightarrow A$ will be judged to be true before $X \rightarrow A$, leading to the pruning of $X \rightarrow A$. This means that $X \rightarrow A$ 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 $Z \rightarrow A$ holds. In other words, all attributes in $X \rightarrow A$ 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 DAFD in the result set satisfies the requirements in Problem 1, we prove the **correctness** and **completeness** of the mining results of DAFDiscover in Theorem 4.5. R1W1 R1D1

Theorem 4.5. *All DAFDs in Σ are valid on $\mathcal{I}_D$, 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 DAFDiscover 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. □

Synthesizing all theorems of Section 3 and Section 4, Theorem 4.6 holds. We note that this is a significant conclusion for direct computation on dirty data that the mining results of the proposed DAFDiscover can accurately reflect the real FDs present in data. Consequently, both DAFDiscover and DAFDiscover+ are effective in mining the dependencies hidden within dirty data.

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

4.4 Analysis of the time and space complexity

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

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

Proof. (1) **Time complexity**. For $\mathcal{I}_D$ with n tuples and m attributes, DAFDiscover's time complexity is $O(n)$ per dependency validation. Compared to Tane, DAFDiscover adds an $O(m+n)$ threshold calculation. In large datasets where $n \gg m$, this becomes $O(n)$, matching Tane's complexity.

Moreover, DAFDiscover uses a more efficient pruning based on Armstrong's axioms, leading to faster search space traversal without increasing overall time complexity beyond Tane. DAFDiscover+ further optimizes threshold computation from $O(m)$ to $O(1)$ and employs a more aggressive pruning, ensuring its complexity remains below both DAFDiscover and 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 needs extra storage for dirty data proportions of m attributes. With Tane's space complexity at $O(\frac{(m+n)2^m}{\sqrt{m}})$, DAFDiscover adds $O(m)$ due to these extra proportions, resulting in a total of $O(\frac{(m+n)2^m}{\sqrt{m}} + m)$. However, as $\sqrt{m}2^m > m$, this simplifies to $O(\frac{(m+n)2^m}{\sqrt{m}})$, matching Tane's upper bound. DAFDiscover+ requires additional space for storing decision thresholds per attribute set, potentially adding $O(2^m)$. However, using a breadth-first traversal akin to Tane, it is optimized to $O(\frac{2^m}{\sqrt{m}})$[16]. Thus, DAFDiscover+'s overall space complexity remains at $O(\frac{(m+n)2^m}{\sqrt{m}})$, again matching Tane's upper bound. □

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 data* 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.

R1W3 R1D4-3 R4D3 To ensure the rigor of the analysis, we introduce two additional assumptions based on those in Section 3.2.1: (*c*) The correctness of an attribute A's values across tuples is independent, and (*d*) the erroneousness of any set of attribute values within a tuple does not reduce the likelihood of another attribute value being erroneous.

We contend that these assumptions are reasonable to some extent. Regarding assumption (*c*), in various data entry and collection scenarios, the occurrence of erroneous data often exhibits randomness and does not systematically affect different tuples. For instance, when entering customer details, an error in one customer's position or phone number typically does not influence the data entry for another customer. Assumption (*d*), on the other hand, is a lenient and easily satisfied condition. Our forthcoming robustness analysis relies on these assumptions. However, we emphasize that the proposed DAFDiscover algorithm does not depend on the dirty dataset $\mathcal{I}_D$ satisfying assumptions (*c*) and (*d*).

We propose a novel metric, DAFD-*prob*, to assess the reliability of DAFD and evaluate the robustness of mining outcomes. By considering both support and validity probability, we demonstrate DAFDiscover produces reliable results for input upper limits with sufficiently high probability.

4.5.2 DAFD-prob: The probability of validity for the mining results. Recall Example 1.1, it underscores that the reliability assessment of the same dataset may vary depending on the error probability of each attribute. Therefore, evaluating the reliability of DAFD necessitates considering the error probability of data sources, and relying solely on the support and confidence of the dataset is insufficient for accurate DAFD reliability evaluation.

Given the properties of data sources, one can compute the validity probability of an FD isomorphic to a DAFD (per Theorem 4.6). A higher validity probability signifies greater reliability of the corresponding FD, while a lower probability indicates reduced reliability. Incorporating data source conditions into this probability calculation makes it a suitable metric for assessing DAFD reliability. Thus, we introduce DAFD-*prob* as a metric. Its calculation requires necessitates scenario-specific data analysis, and we outline a method for determining DAFD-*prob* based on the assumptions presented in this paper, offering a theoretical lower bound expression.

DAFD-*prob*. According to Theorem 3.3, if there exists an upper bound $k = a_i n$ for erroneous tuples under an attribute, there is also an upper bound $\tau = \sum_{A_i \in X \cup Y} a_i n$ for erroneous tuples (denoted as event E_A) in any DAFD candidate. In this case, if a DAFD is valid on $\mathcal{I}_D$, its isomorphic FD on the underlying clean data $\mathcal{I}$ is also valid. For an FD candidate ψ_{FD} and its corresponding isomorphic DAFD candidate φ, it has $DAFD\text{-}prob = \Pr(\models \psi_{FD}) \geq \Pr(\models \psi_{FD} | E_A and \models \varphi) \cdot \Pr(\models \varphi | E_A) \cdot \Pr(E_A)$. By Theorem 4.6, $\Pr(\models \psi_{FD} | E_A and \models \varphi) = 1$. If DAFDiscover confirms $\mathcal{I}_D \models \varphi$, then $\Pr(\models \varphi | E_A) = 1$, leading to $\Pr(\models \psi_{FD}) \geq \Pr(E_A)$.

We proceed to compute $\Pr(E_A)$. For $X \rightarrow Y$, let p denote the probability that a tuple being dirty in the projection of the dataset onto $X \cup Y$. 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 has $\Pr(B \text{ clean}) \geq \Pr(B \text{ clean} | A is \text{ dirty})$, which implies $\Pr(B \text{ clean}) \leq \Pr(B \text{ clean} | A \text{ clean})$. Further, 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 is either accurate or inaccurate, and each tuple can only contain erroneous data or not. It dictates a binomial distribution for erroneous tuples. Specifically, for dirty data $\mathcal{I}_D$, the number of erroneous tuples τ_0 follows $\tau_0 \sim B(n, p)$, where n is the total tuples. As $n \rightarrow +\infty$, the Central Limit Theorem allows us to approximate this binomial distribution with a normal distribution $\tau_0 \sim N(np, np(1-p))$.

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	10^3	10^3	10^3
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

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:

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

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 establishes the lower bound of the reliability of the mining results, assuming a constant number of dependency violations in the dirty dataset. However, data characteristics should factor into reliability assessments. People often trust dependencies with fewer violations more, even in dirty datasets. Thus, reliability evaluations should consider both the *support* and *validity probability* perspectives.

Consider a DAFD φ in the result set Σ with support denoted as *sup*. The upper limit of erroneous data proportion for attribute A_i is set based on the 3σ principle [36]. From the support viewpoint, it is evident that the support of φ satisfies: $sup \geq 1 - \frac{\sum_{A_i \in X \cup Y}(a_i n + 3\sqrt{na_i(1-a_i)})}{n}$, where $X \cup Y \subseteq Attr(\mathcal{R})$, and thus $|X \cup Y| \leq m$. Therefore, when m is small 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, leading to a sufficiently large *sup* for φ. Table 3 illustrates the relation between m, a_i, and *sup*. It highlights a relatively small m, sufficiently small a_i, and a sufficiently large n yield a large support for φ. Thus, 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.

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

$$\text{DAFD-}prob = \frac{1}{2} + \int_{np_0}^{np_0 + \frac{(\tau-np)}{\sqrt{np(1-p)}}\sqrt{np_0(1-p_0)}} f_0(x)dx \tag{2}$$
$$\geq \frac{1}{2} + \int_{np_0}^{np_0 + 3\sqrt{np_0(1-p_0)}} f_0(x)dx \geq 0.997 \tag{3}$$

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

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

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

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

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

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

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

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

Table 4: Summary of datasets

Dataset	#*Tuples*	#*Attributes*	#*FDs*	*Attributes*
Abalone[28]	4177	9	137	[Sex, Length, Diameter, Height, Whole_weight, Shuck_weight, Viscera_weight, Shell_weight, Rings]
Chess[4]	28056	7	1	[White King file, White King rank, White Rook file, White Rook rank, Black King file, Black King rank, optimal depth-of-win]
Breast-cancer[38]	699	11	46	[Sample code number, Clump Thickness, Uniformity of Cell Size, Uniformity of Cell Shape, Marginal Adhesion, Single Epithelial Cell Size, Bare Nuclei, Bland Chromatin, Normal Nucleoli, Mitoses, Class]
Forestfires[12]	517	13	442	[X, Y, month, day, FFMC, DMC, DC, ISI, temp, RH, wind, rain, area]
Air-quality[34]	9358	15	1765	[CO(GT) , PT08.S1(CO), NMHC(GT), C_6H_6(GT), PT08.S2(NMHC), NO_x(GT), PT08.S3(NO_x), NO_2(GT), PT08.S4(NO_2), PT08.S5(O_3), T, RH, AH, temperature_suitability, humidity_comfort_level]
Bike-sharing[14]	731	16	519	[instant, dteday, season, yr, mnth, holiday, weekday, workingday, weathersit, temp, atemp, hum, windspeed, casual, registered, cnt, quarter]
Diabetic[27]	2278	10	55	[SEQN, age_group, RIDAGEYR, RIAGENDR, PAQ605, BMXBMI, LBXGLU, DIQ010, LBXGLT, LBXIN]
Raisin[10]	900	8	43	[Area, MajorAxisLength, MinorAxisLength, Eccentricity, ConvexArea, Extent, Perimeter, Class]
Bitcoinheist[1]	2916697	10	-	[address, year, day, length , weight, count, looped, neighbors, income, label]
Caulkins[11]	1685	12	227	[Year, Locat'n, Drug, Drg Cde, Q in Grams, Low, High, Low_Pur, Hi_Pur, Avg, Ab_Pur, Pure_Qty]
Hughes[11]	400	8	3	[h1, hr, dtime, dstatus, grp, test0, result0, confirm0]

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

For a high-probability input upper bound, DAFDiscover yields mining results with high validity probability. This confirms the reliability of DAFDiscover's mining results and demonstrates the robustness of our proposed DAFD discovery solution.

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 eleven 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 DAFDiscover(+) from this paper and compare it with the following baseline discovery methods: • Tane, a classical schema-driven AFD mining algorithm [16]. • Pyro: A SOTA AFD mining method that traverses the entire search space through ascend and trick-down phases, with stripped partitions to mine all AFDs [20]. • HyFD: A hybrid and full-scale FD mining algorithm [30]. • FDx: An FD mining method based on structure learning for noisy data [41]. • MiDD: [25] 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. • SoFD: A naive mining algorithm for soft functional dependencies[18].

Metrics. Algorithm performance is evaluated using three metrics: (1) Mining result quality, 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 of discovered dependencies. (2) Scalability, gauged by execution *time* and maximum *memory*. (3) The accuracy of DAFDs in reflecting actual FDs in error-prone data, evaluated by DAFD-*prob* and support.

Implementation. Considering the absence of standard datasets for evaluating dependencies on dirty data, we enhance real datasets by duplicating them (i.e., data×128) and adopt the noise injection method of adding "*" from Bart[3] to introduce errors. Noise is introduced into each attribute based on a set upper limit or probability (noi%), simulating real-world data errors. We provide comprehensive parametric information on noise injection in our extended paper.

It is crucial to mention that for metric (1) calculation, the golden standard for AFD and DAFD candidates is the mining results from clean datasets, considered as all valid FDs for dirty datasets. Specifically, HyFD and Pyro use their respective clean dataset mining outcomes as a comparison baseline, while FDx lacks a suitable baseline, thus omitting P, R, and $F1$ calculations. For other algorithms, the Tane FD mining results on clean datasets serve as the baseline. R4W1 R4W2

5.2 Overall performance evaluation

Experiment Setup: Test datasets were generated using the following datasets: Breast-cancer, Chess, Forestfires, Abalone, and Raisin. The test data generation process involved initially selecting a small subset of tuples from the original datasets. These subsets underwent amplification, increasing their size by at least 300 times, resulting in clean, amplified datasets. Subsequently, noise was injected into these amplified datasets, adhering to predefined noise ratio limits. The final test datasets, along with their specific details, are presented in Table 8, with the first column depicting the generated datasets' information. R2W3 R2D3

It demonstrates 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.

5.3 Robustness verification of DAFDiscover

Experiment Setup: The Breast-cancer, Cauklins, and Hughes datasets were selected for evaluation. For each dataset, specific attributes of every tuple were altered to noise data based on a predefined probability (not an upper limit). Additionally, for the remaining attributes, each tuple's value was changed to noise data

Table 5: Overall performance comparison

Methods	*Breast-cancer* (7K)					*Chess* (10K)					*Forestfires* (6K)					*Abalone* (40K)					*Raisin* (10K)				
	P	*R*	*F1*	*Num*	**Time**	*P*	*R*	*F1*	*Num*	**Time**	*P*	*R*	*F1*	*Num*	**Time**	*P*	*R*	*F1*	*Num*	**Time**	*P*	*R*	*F1*	*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
MtDD	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

Table 6: Robustness verification (with noise injection)

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: Performance comparison on dirty datasets that not satisfies the assumptions

	Chess (×15)				*Raisin* (×15)				*Bike-sharing* (×15)			
	P	*R*	*F1*	*Num*	*P*	*R*	*F1*	Num	*P*	*R*	*F1*	*Num*
DAFDiscover	1	1	1	1	0.88	1	0.93	49	0.77	0.57	0.65	111
DAFDiscover+	1	1	1	1	0.88	1	0.93	49	0.77	0.57	0.65	111
Tane	0.5	1	0.67	2	0.69	0.79	0.74	49	0.58	0.31	0.40	80
MtDD	0	0	-	2	0.88	1	0.93	49	0.22	0.11	0.14	72
SoFD	0.5	1	0.67	2	0.88	1	0.93	49	0.37	0.15	0.22	63
	Chess (×20)				*Raisin* (×20)				*Bike-sharing* (×20)			
DAFDiscover	0.5	1	0.67	2	0.88	1	0.93	49	0.73	0.52	0.61	107
DAFDiscover+	0.5	1	0.67	2	0.88	1	0.93	49	0.73	0.52	0.61	107
Tane	0.5	1	0.67	2	0.69	0.79	0.74	49	0.54	0.28	0.37	78
MtDD	0	0	-	1	0.88	1	0.93	49	0.22	0.11	0.14	72
SoFD	0.5	1	0.67	2	0.88	1	0.93	49	0.37	0.15	0.22	63
	Chess (×25)				*Raisin* (×25)				*Bike-sharing* (×25)			
DAFDiscover	1	1	1	1	0.88	1	0.93	49	0.71	0.50	0.59	106
DAFDiscover+	1	1	1	1	0.88	1	0.93	49	0.71	0.50	0.59	106
Tane	0.5	1	0.67	2	0.69	0.79	0.74	49	0.70	0.34	0.46	73
MtDD	0	0	-	1	0.88	1	0.93	49	0.22	0.11	0.14	72
SoFD	0.5	1	0.67	2	0.88	1	0.93	49	0.71	0.39	0.50	82

with a probability of 10^{-6}. Details of the noise generation for each dataset are presented in Table 11

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.

R1D4-3 R4D3 To further validate the robustness, we conduct tests on dirty datasets that do not satisfy the assumptions outlined in this paper. These tests focused primarily on assessing algorithm performance under scenarios of *amplified attribute error rates* and the introduction of *dependent errors*. Specifically, compared to the experimental settings in Section 5.2, we extend the noise levels to 15, 20, and 25 times for three datasets. Additionally, some non-independent noises are introduced during the noise injection. The experimental results are reported in Table 7. DAFDiscover(+) still outperforms other methods overall. Regarding the mining results on Raisin, our analysis suggests that due to the presence of numerous key attributes, changes in noise injection effects are not significant without substantial variations in noise injection levels. Consequently, several algorithms exhibit similar mining performance across different noise injection levels. On Bike-sharing dataset, although high noise rates affect the absolute performance of DAFDiscover(+), its advantage over other algorithms remains evident.

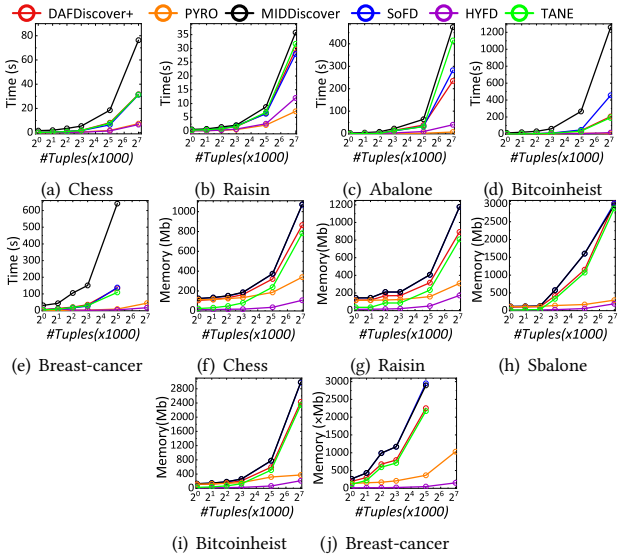

Figure 3: Scalability evaluation with varying data amounts

5.4 Scalability evaluation with varying tuples

Experiment Setup: To evaluate the temporal and spatial performance of DAFDiscover in relation to varying tuple counts, the following datasets were selected: Chess, Abalone, Raisin, Bitcoinheist, and Breast-cancer. Each dataset underwent amplification to obtain a sufficient number of tuples, with noise injected into every attribute up to a specified limit (Chess amplified from 8000, Abalone from 4000, Bitcoinheist remained unamplified, while Raisin and Breast-cancer were amplified below 1000). Subsequently, the time and memory costs of the algorithm were measured at tuple counts of 1000, 2000, 4000, 8000, 32000, and 128000. The details of noise injection are presented in Table 12.

We evaluate the scalability performance on five datasets with data amplification, as the results depicted in Figure 3. 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,

Table 8: Experiment setup for Section 5.2

Dataset	from	#column	Amplification factor	Upper limit vector for injected noise ratio
Breast-cancer-enhanced-dirty	***Breast-cancer***	11	700	[0,0,0,0,0,0.001,0,0.005]
Chess-enhanced-dirty	***Chess***	7	10000	[0,0,0,0,0.002,0,0.005]
Forestfires-enhanced-dirty	***Forestfires***	13	300	[0,0,0,0,0.001,0,0,0.003,0,0,0,0,0.001]
Abalone-enhanced-dirty	***Abalone***	9	2000	[0,0,0,0,0.001,0,0,0.003,0]
Raisin-enhanced-dirty	***Raisin***	8	1000	[0,0,0,0,0.006,0,0.007,0.001]

Table 9: 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

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 shows a decrease in runtime with increasing tuples, attributed to fewer iterations in mining resulting from dependency variations. The absolute time and space costs of HyFD and Pyro are also lower. While DAFDiscover+ has higher consumption than HyFD and Pyro due to the added computational cost of calculating dynamic 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 9, 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 10 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 because the constant ϵ used in AFD is significantly higher than the upper limit of erroneous data for humiditycomfortlevel. 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 relationships indicates that the structure learning strategy alone cannot fully address these limitations inherent in static threshold-based approaches.

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.

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

Table 10: 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)	√	√				√	

Table 11: Experiment setup for Section 5.3

Dataset	Upper limit vector for injected noise ratio
Breast-cancer	$[10^{-6}, 10^{-6}, 10^{-6}, 10^{-6}, 10^{-6}, 10^{-6}, 10^{-6}, 10^{-6}, 0.005, 10^{-6}, 0.001]$
Caulkins	$[10^{-6}, 10^{-6}, 10^{-6}, 10^{-6}, 0.01, 10^{-6}, 10^{-6}, 10^{-6}, 0.005, 10^{-6}, 0.001, 10^{-6}]$
Hughes	$[10^{-6}, 10^{-6}, 10^{-6}, 0.005, 10^{-6}, 0.02, 0.01, 0.005]$

Table 12: Experiment setup for Section 5.4

Dataset	Upper limit vector for injected noise ratio
Breast-cancer	[0,0,0,0,0,0,0,0,0.005,0,0.001]
Abalone	[0,0,0,0,0,0,0.001,0,0.005]
Chess	[0,0,0,0,0.002,0.005,0]
Raisin	[0,0,0,0,0.006,0,0.007,0.001]
Bitcoinheist	[0,0,0,0,0.002,0,0,0,0.003,0,0.001]

[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] Loredana Caruccio, Vincenzo Deufemia, and Giuseppe Polese. 2020. Mining relaxed functional dependencies from data. *Data Min. Knowl. Discov.* 34, 2 (2020), 443–477.

[9] 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

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

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

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

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

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

[15] 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

[16] 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

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

[18] 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.

[19] Aimad Karkouch, Hajar Mousannif, Hassan Al Moatassime, and Thomas Noël. 2016. Data quality in internet of things: A state-of-the-art survey. *J. Netw. Comput. Appl.* 73 (2016), 57–81.

[20] 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

[21] 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

[22] 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

[23] 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, Vol. 1777)*, Carlo Zaniolo, Peter C. Lockemann, Marc H. Scholl, and Torsten Grust (Eds.). Springer, 350–364. https://doi.org/10.1007/3-540-46439-5_24

[24] 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

[25] 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

[26] 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

[27] 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.

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

[29] 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, Vol. 1973)*, Jan Van den Bussche and Victor Vianu (Eds.). Springer, 189–203. https://doi.org/10.1007/3-540-44503-X_13

[30] 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

[31] 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

[32] 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

[33] Shaoxu Song and Aoqian Zhang. 2020. IoT Data Quality. In *the 29th ACM International Conference on Information and Knowledge Management CIKM.* 3517–3518.

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

[35] 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

[36] Xi Wang and Chen Wang. 2020. Time Series Data Cleaning: A Survey. *IEEE Access* 8 (2020), 1866–1881.

[37] 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

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

[39] 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

[40] 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

[41] 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