Discovering Approximate Denial Constraints in Large Databases

Albert Martin
Universitat Politècnica de Catalunya
Barcelona, Spain
albert.martin.g@upc.edu

Eduardo C. de Almeida
Federal University of Paraná
Curitiba, Brazil
eduardo@inf.ufpr.br

Oscar Romero
Universitat Politècnica de Catalunya
Barcelona, Spain
oscar.romero@upc.edu

Anna Queralt
Universitat Politècnica de Catalunya
Barcelona, Spain
anna.queralt@upc.edu

ABSTRACT

Denial Constraints (DCs) form a highly expressive integrity rule language that subsumes many used formalisms such as keys and functional dependencies, making them widely adopted in applications that require the manipulation of rich sets of data constraints. This expressiveness has motivated the development of numerous algorithms for automatically discovering DCs from data, with particular emphasis on the discovery of approximate DCs to improve robustness to erroneous data. However, existing DC discovery algorithms exhibit computational costs that are quadratic in the number of tuples and exponential in the number of attributes, and most cannot accommodate changes in the data. Moreover, they often produce thousands of uninformative DCs. These limitations make current DC discovery algorithms difficult to use effectively on very large and dynamic databases. In this paper, we present LIMA, an approximate DC discovery algorithm that efficiently discovers DCs on very large and dynamic databases. LIMA uses statistical methods to infer properties of DCs from reduced samples, and introduces a novel discovery framework that exploits a more restrictive definition of DC validity to substantially reduce the cost of searching for valid DCs. We experimentally demonstrate that LIMA achieves significantly better scalability than current algorithms with respect to both rows and attributes, while also discovering higher-quality sets of DCs with precisions several orders of magnitude higher than the state of the art, both in static and in dynamic datasets.

PVLDB Reference Format:
Albert Martin, Eduardo C. de Almeida, Oscar Romero, and Anna Queralt. Discovering Approximate Denial Constraints in Large Databases. PVLDB, 19(9): 2086 - 2098, 2026.
doi:10.14778/3819518.3819536

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

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 19, No. 9 ISSN 2150-8097.
doi:10.14778/3819518.3819536

1 INTRODUCTION

Integrity Rules (IRs) are a fundamental form of metadata in databases, as they formally describe semantic relationships among attributes that must hold in valid data instances. Classical examples include Functional Dependencies (FDs), Order Dependencies (ODs), Unique Column Combinations (UCCs), and Conditional Functional Dependencies (CFDs). The expressive power of an IR language is determined by the types of relationships it can specify, as no single formalism is sufficient to capture all relevant semantic dependencies. Denial Constraints (DCs) were introduced as a unifying framework that generalizes a wide range of IR languages [1, 2]. Informally, a DC φ states the impossibility of a set of predicates being simultaneously true for any pair of tuples in the data. The expressiveness of DCs stems from the flexibility in defining their predicates, allowing them to subsume constraints, such as FDs, UCC, and ODs, as illustrated in Table 1 and introduced more formally in Section 2. For example, DCs can express complex rules like "no two tuples share the same Social Security Number (SSN)" or "it is forbidden for one tuple to have a lower salary but a higher tax than another tuple". DCs can represent many relationships beyond those expressible in any particular IR language, making them suitable for applications that require reasoning over rich sets of IRs, including data cleaning [1], integration [3], database design [12], and query optimization [20].

The discovery of DCs from raw data has attracted significant attention due to the great value of identifying the DCs intrinsic to a dataset. Existing algorithms typically follow a three-stage pipeline: (1) build the predicate space, (2) build a data structure, called evidence set, mapping predicate sets to supporting tuple pairs, and (3) enumerate the DCs traversing the space of candidate DCs using the evidence set [5, 7, 9, 14–19]. While these algorithms are effective on small datasets, they face severe scalability challenges: the cost of building the evidence-set grows quadratically with the number of tuples [5, 7, 15, 16], and DC enumeration grows exponentially with the number of attributes [5, 7, 15, 16, 19]. Most of these algorithms assume static datasets and evaluate the validity of DCs using point estimates, providing limited reasoning about uncertainty and DC persistence as datasets evolve. As a result, state-of-the-art algorithms become impractical for dynamic datasets, and for static datasets exceeding a million rows or twenty attributes. Furthermore, several studies highlight the limited quality of discovered DCs, as algorithms often generate thousands of constraints that do not correspond to meaningful rules in the data [7, 9, 10, 15, 16].

We tackle all aforementioned scalability and quality challenges by completely redesigning the DC discovery process, from the methods used to determine which DCs should be or not be discovered to the framework used to search through the space of candidate DCs. Specifically, we introduce the following contributions:

- First, we present an statistical inference method to estimate the distribution of joint predicate probabilities from a reduced sample of tuple pairs, offering improved reliability than current methods.
- Second, we introduce a new structure for DC discovery that departs entirely from the traditional three-stage pipeline, named Lattice Information Maximization Algorithm (LIMA). LIMA uses the aforementioned statistical method to make decisions about candidate DCs without building full evidence sets, significantly reducing the computational cost of DC discovery. LIMA iteratively refines its knowledge of the joint probability distributions of predicate sets by concentrating computation on the operations expected to yield the highest information gain.
- Third, at any point in time, LIMA also exploits the statistical method to determine which predicate sets are supported by sufficient statistical evidence to be considered valid DCs for the given data. As a result, LIMA is suitable not only for static DC discovery, but also for dynamic DC discovery, since updates to the data naturally translate into already supported updates of the joint probability distributions.

We experimentally validate the improved scalability of LIMA compared to the state of the art algorithms. Our results show that LIMA scales significantly better in both execution time and memory footprint for datasets with large numbers of rows and columns, making it well suited for the discovery of approximate DCs on very large databases. We also demonstrate that, by adopting a more restrictive definition of DC validity [10], the quality of the DCs discovered by LIMA is substantially improved, achieving precisions several orders of magnitude higher than the state of the art with a negligible impact on recall. Finally, we highlight LIMA's ability to update the set of valid DCs as data evolves, showing that the algorithm is also applicable to DC discovery in dynamic environments with performances superior than the state of the art.

The remainder of this paper is organized as follows: Section 2 reviews DCs, the discovery problem, and related work. Section 3 introduces our statistical framework for modeling DC validity. Section 4 presents LIMA, detailing its lattice structure, scheduling engine, and sampling strategies. Section 5 reports experimental results on scalability, quality, and incremental discovery. Section 6 concludes and presents future directions.

2 BACKGROUND

This section provides a comprehensive overview of Denial Constraints, including their formal definition and the formulation of the DC discovery problem. It also presents a review of existing algorithms proposed in the literature that aim to solve this problem, highlighting their key approaches, strengths, and limitations.

2.1 DCs

Denial Constraints (DCs) express the exclusivity of certain combinations of predicates over pairs of distinct tuples (t_x, t_y) from a relation R. A predicate P is any boolean function defined over these tuple pairs. Formally, a DC φ is expressed as:

$$\forall t_x, t_y \in R, \neg(P_1 \wedge P_2 \wedge \cdots \wedge P_k)$$

Language	IR	DC Representation
UCC	$\{SSD\}$	$\neg(t_x.SSD = t_y.SSD)$
UCC	$\{Client, Product\}$	$\neg(t_x.Client = t_y.Client \wedge t_x.Product = t_y.Product)$
FD	$City \to State$	$\neg(t_x.City = t_y.City \wedge t_x.State \neq t_y.State)$
OD	$\{Tax, Salary\}$	$\neg(t_x.Salary < t_y.Salary \wedge t_x.Tax > t_y.Tax)$

Table 1: Examples of integrity rule languages representable by DCs.

This formalism is highly expressive, capable of representing a wide variety of integrity rules. Table 1 illustrates how different types of common integrity constraints can be represented using DCs. Following the literature [5, 7, 9, 15, 16, 19], we consider predicates taking the form $P(t_x, t_y) = t_x.A\phi t_y.B$, with ϕ being an operator among the set $\{=, \neq, \geq, \leq, >, <\}$ and A and B attributes of relation D. This greatly reduces the complexity of the language whilst maintaining its ability to represent widely used Integrity Rule languages, including but not limited to Keys, Functional Dependencies, Conditional Functional Dependencies or Order Dependencies.

2.2 DC Discovery Problem

The space of constraints that can be expressed using DCs is extremely large. The objective of the DC discovery problem is to identify the subset of this space that corresponds to actual rules in the data. To this end, the first work on DC discovery introduced a set of rules to determine when a DC $\varphi = \neg(P_1 \wedge P_2 \wedge \cdots \wedge P_k)$ is considered valid [7].

The most fundamental property of any valid DC is that it must hold on the data. The first DC validity rule ensures this property:

- **Satisfied**: Let $p(\varphi)$ be the proportion of tuple pairs that satisfy all predicates in φ, and thus violate the DC. A DC is satisfied when $p(\varphi) = 0$. This condition can be relaxed to allow approximate DCs by introducing an approximation factor ϵ, and considering a DC satisfied when $p(\varphi) < \epsilon$.

However, not all satisfied DCs correspond to meaningful rules in the data. It is possible to add any number of additional predicates to a satisfied DC and still obtain a new, satisfied DC. To prevent the discovery of an exponentially large number of such derivative DCs, a second rule was introduced:

- **Minimal**: Removing any predicate from φ results in a DC that is no longer satisfied.

Even if a DC is both satisfied and minimal, it may still fail to represent a real-world constraint. For example, tautological forms like $\neg(P \wedge \neg P)$ are always satisfied regardless of the data. To filter out such cases, a third rule was added:

- **Non-trivial**: The DC is not satisfied by every possible dataset.

These three rules form the original definition of DC validity used by all existing DC discovery algorithms. However, this definition

still accepts many DCs that do not correspond to meaningful constraints of the data [7, 9, 10, 15]. To address this, an additional rule has been introduced to further refine the notion of validity [10]:

- **Sound**: The satisfaction of the DC is not artificially achieved through independent predicates.

This final rule ensures that valid DCs are satisfied because they capture genuine relationships in the data, instead of being agglomerations of independent predicates that hold by chance.

The goal of the DC discovery problem is to enumerate all DCs that satisfy the DC validity definition. Traditionally, this meant reporting all DCs that are satisfied, minimal, and non-trivial. Given the improved quality of results when incorporating the final rule, our algorithm is designed to also exclude DCs that are not sound.

2.3 DC Discovery Overview

All static DC discovery algorithms published before follow a common three-stage structure [5, 7, 9, 14–16, 19]:

Predicate Space Building. Given a dataset R with attributes $A_1, A_2, \dots, A_l$, this stage aims to enumerate all possible predicates that may form the discovered DCs, known as the predicate space P. The computational cost of this stage is typically negligible, as it mainly involves iterating over all pairs of columns sharing the same domain and generating predicates according to their data types.

Evidence Set Building. Given the predicate space P, the goal of this stage is to construct the evidence set. The evidence set is a data structure that stores, for every subset of predicates $P' \subseteq P$, the number of tuple pairs $(t_x, t_y) \in R \times R$ that satisfy exactly those predicates. All prior studies that include experimental analyses of DC discovery consistently highlight this step as the most computationally expensive [5, 7, 15, 16]. These experiments show how the cost of building the evidence set tends to be quadratic over the number of tuples in the data, reaching up to an hour of execution time for datasets of over a million tuples. This makes it difficult for these algorithms to scale to large databases.

Denial Constraint Enumeration. Given the evidence set, this stage aims to enumerate all sets of predicates that form valid DCs over R. The enumeration consists on traversing the space of candidate DCs $\varphi \in 2^P$, using the evidence set to efficiently compute $p(\varphi)$. If the joint probability of a set of predicates does not exceed the approximation factor ϵ, the corresponding DC is accepted, and the search space is pruned to prevent the generation of non-minimal DCs. Several traversal strategies have been proposed [5, 7, 9], though their differences in performance are generally minor [16]. Although evidence set construction typically dominates the total runtime, experimental results have shown that the time required to enumerate DCs from the evidence set still grows exponentially with the number of columns in the dataset [5, 7, 15, 16, 19], quickly surpassing an hour on our experiments with over 20 attributes in Section 5. As before, this exponentially increasing cost with respect to the number of attributes makes it difficult for these algorithms to scale to large databases.

2.4 Related Work

Static DC Discovery: The first algorithm designed to solve the DC discovery problem was FastDC [7]. It performs a depth-first search over the space of candidate DCs, halting at satisfied DCs to ensure minimality. To compute the exact number of tuple pairs that satisfy a given set of predicates φ, FastDC introduced the evidence set, described in Section 2.3. The efficiency of this design was first improved in BFastDC [14] by operating with bit operations. A major improvement was introduced by DCFinder [15] through the use of Position List Indices (PLIs). These data structures store, for each distinct value in a given attribute, the tuples that have this value. PLIs significantly accelerate the construction of the evidence set by making it trivial to identify the tuple pairs that satisfy a given, highly selective, predicate. Further enhancements were proposed in [16] with the introduction of the Evidence Context Pipeline (ECP). This method allows the identification of the exact set of tuple pairs that satisfy a given combination of predicates without the need to reference each pair individually, greatly speeding up the construction of the evidence set.

A different approach to the DFS-based DC enumeration used by the previous algorithms was explored in Hydra [5], named Evidence Inversion. This algorithm maintains a set of candidate exact DCs, which is expanded as new evidence implies they are not satisfied. While this algorithm is designed exclusively for exact DC discovery, it was generalized for approximate discovery in FastADC [19]. The study of alternative approximation functions to quantify DC satisfaction was studied in [4], in addition to proposing ADCMiner, which enumerates DCs using the MMCS algorithm [11].

Incremental DC Discovery: DC discovery algorithms were originally designed to discover DCs from static datasets. However, real-world datasets are often dynamic, with both the data and its underlying relationships evolving over time. The first work to address incremental DC discovery was presented in [18], introducing dynamic indices that can be updated as new data arrives, allowing the tracking of modifications to the set of valid DCs. Another algorithm designed for dynamic datasets was proposed in [17], which instead tracks and corrects changes to the evidence set using evidence contexts from [16].

DC Quality: While individual algorithms introduce novel optimizations to accelerate the search, they all share the same objective: to enumerate all DCs that are satisfied, minimal, and non-trivial. As a result, the quality of the discovered constraints has remained largely consistent across algorithms. Many publications have noted the limited quality of the results produced by these methods [4, 7, 10, 15], with some explicitly stating that improving the precision of DC discovery is necessary for these algorithms to reach their full potential [10, 19]. To address this, an additional rule for DC validity, called Soundness, was introduced in [10] to greatly improve the precision of the discovered DCs.

3 MODELING DC VALIDITY

As described in Section 2.2, we consider a DC to be valid if it is satisfied, minimal, nontrivial, and sound. Given a set of predicates φ forming a DC, its validity depends solely on the joint probability of all predicates being true simultaneously, $p(\varphi)$. Traditional DC discovery systems estimate $p(\varphi)$ using empirical violation counters derived from the evidence set, as discussed in Section 2.3. While straightforward, these approaches do not capture the full information about $p(\varphi)$ present in the data in an efficient manner. First, they require evaluating all predicates in φ on every pair of tuples,

an operation whose impact on the efficiency of DC discovery is discussed in Section 4. More importantly, relying on a point estimate for $p(\varphi)$ discards information about the confidence in this estimate, as well as the likelihood of alternative values of the probability.

In this section, we present a fundamentally novel approach to determining DC validity, representing a significant departure from the point-estimate-based methods used in the state of the art. We develop a statistical analysis method to model the distribution of $p(\varphi)$, enabling more informed decisions than those obtained from a simple point estimate of this probability. Building on this model, we develop statistical tests to determine whether a DC is satisfied and sound based solely on the evaluation of φ over an arbitrarily sized sample of tuple pairs. These novel proposals serve as a more robust and efficient framework for determining DC validity.

3.1 Statistical inference of $\mathbf{p}(\varphi)$

We treat $\mathbf{p}(\varphi)$ as a random binary variable and model its evaluation on a random tuple pair $\varphi(t_x, t_y)$ as a Bernoulli experiment [10]. Consequently, evaluating φ over n random, uniformly drawn tuple pairs results in k pairs satisfying all predicates, distributed as $k \sim Binomial(n, p(\varphi))$, where the likelihood of observing exactly k successes out of n trials is:

$$\binom{n}{k} p(\varphi)^k (1 - p(\varphi))^{n-k}$$

This expression is usually read as the likelihood of obtaining k successes out of n independent trials, given probability $p(\varphi)$. Conversely, it can also be interpreted as the likelihood of $p(\varphi)$ taking a particular value, given that k of n tuple pairs satisfy φ. The $p(\varphi)$ that maximizes this likelihood is $\frac{k}{n}$, which is the estimate for the joint probability $p(\varphi)$ used by current methods, with the drawbacks described above. This estimation method ignores the likelihood of alternative values for $p(\varphi)$, which directly reflects the confidence in the estimated probability. By normalizing the likelihood over all possible values of $p(\varphi)$, we obtain a probability density function for $p(\varphi)$ conditioned on the evaluation of φ over n tuple pairs:

$$p(\varphi) \sim Beta(k + 1, n - k + 1) \quad (1)$$

This probability distribution for φ captures richer information than a simple maximum-likelihood point estimate because it also reflects the uncertainty around its value. This additional information leads to more predictable discovery behavior and allows DC discovery systems to operate effectively even under sampling or in heterogeneous datasets.

Directly comparing probabilities becomes numerically unstable when $p(\varphi)$ is very small, which is common in DC discovery where predicates sets tend to be highly selective. For numerical stability and precision, we instead work with $ln(p(\varphi))$. The distribution of this logarithm has known moments, uniquely determined by n and k, and can be accurately approximated as:

$$ln(p(\varphi)) \sim Normal(\mu, \sigma^2)$$

With:

$$\mu = \psi(k + 1) - \psi(n + 2)$$
$$\sigma^2 = \psi_1(k + 1) - \psi_1(n + 2)$$

Where ψ and ψ_1 are the digamma and trigamma functions [8, 10].

3.2 Determining DC satisfaction

As described in Section 2.2, a DC φ is satisfied if and only if:

$$p(\varphi) \leq \epsilon$$

That is, a DC φ is satisfied if the proportion of tuple pairs that violate it (i.e., satisfy all predicates in φ) is below the approximation factor ϵ. Current methods verify this by evaluating φ on the entire dataset and checking whether the estimated $p(\varphi)$ falls below ϵ. In contrast, leveraging the distribution of $p(\varphi)$ allows us to assess satisfaction without exhaustively evaluating all tuple pairs. For any number of evaluated tuple pairs, we compute an upper bound for $ln(p(\varphi))$ with confidence level α, as:

$$\mu + \sigma z_\alpha$$

Where z_α is the (α) quantile of the standard normal distribution. If $\ln(\epsilon) > \mu + \sigma z_\alpha$, then with confidence level α, we conclude that $p(\varphi) \leq \epsilon$, and therefore the DC φ is satisfied.

3.3 Determining DC soundness

A similar reasoning applies when determining whether a DC is sound. As defined in [10], a DC φ is sound if and only if:

$$p(P| \varphi \setminus \{P\}) < p(P| \varphi' \setminus \{P\}) \quad \forall \varphi' \subset \varphi, \forall P \in \varphi'$$

Each of these inequalities can be rewritten as a ratio:

$$\frac{p(P| \varphi \setminus \{P\})}{p(P| \varphi' \setminus \{P\})} < 1$$

Taking logarithms gives a linearized form:

$$ln(p(P| \varphi \setminus \{P\})) - p(P| \varphi' \setminus \{P\}) < 0$$

Using the identity $p(P \mid \varphi \setminus P) \cdot p(\varphi \setminus P) = p(\varphi)$, this condition can be expressed in terms of joint probabilities of predicate sets:

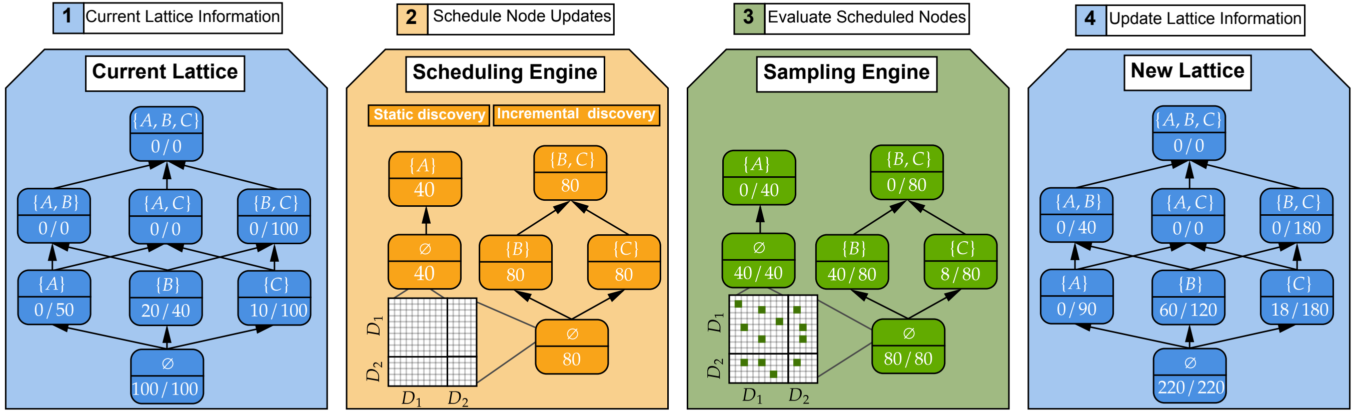
$$ln(p(\varphi)) - ln(p(\varphi \setminus \{P\})) - ln(p(\varphi')) + ln(p(\varphi' \setminus \{P\})) < 0$$

Since we know the distribution of each probability term, we can derive upper bounds for the full expression as done in the previous section. If the upper bound is strictly below 0 for all inequalities, the DC is sound.

4 LATTICE INFORMATION MAXIMIZATION ALGORITHM (LIMA)

As discussed in Section 2.3 and corroborated by our experiments in Section 5, existing DC discovery algorithms do not scale to databases with more than a million rows or more than 20 attributes. Additionally, most of them are unable to update their results as new data is ingested or updated. More importantly, a large fraction of the discovered DCs fail to capture meaningful semantic relationships in the data, and instead are an agglomeration of independent predicates that happen to hold in a specific dataset [10]. In light of this, we propose a novel algorithm named Lattice Information Maximization Algorithm (LIMA) for discovering approximate DCs that addresses these challenges:

- **Quality**: Our algorithm ensures that discovered DCs are sound [10], guaranteeing that they capture statistically significant relationships among predicates.



(a) Lattice state displaying the number of tuple pairs fulfilling all predicates in φ, $K(\varphi)$, out of a sample of size N, $N(\varphi)$, used to statistically infer DC validity for each $\varphi \in \Omega$.

(b) Schedule indicating by how many tuple pairs $N'(\varphi)$ each sample must increase and their source, both chosen to maximize information gain and preserve statistical relevance.

(c) Sampled schedule from Figure 1b with the number of tuple pairs $K'(\varphi)$ satisfying each $\varphi \in \Omega$ out of the scheduled $N'(\varphi)$, sampled from the scheduled source.

(d) Lattice state displaying the new values of $K(\varphi)$ and $N(\varphi)$ for each $\varphi \in \Omega$, obtained by adding the new information from the sampled schedule $K'(\varphi)$ and $N'(\varphi)$.

Figure 1: LIMA, a novel statistical-based algorithm for DC discovery. The diagram summarizes the role of each component within the core loop of the algorithm. The main data structure, the lattice, is iteratively updated to maximize the information available about the distributions of the joint probabilities of predicate sets. The scheduling engine selects the updates that are expected to yield the greatest information gain, and the sampling engine executes these updates.

- **Row Scalability**: Our algorithm operates on a representative sample of the data. Unlike the sampling scheme proposed in [9], which samples individual tuples, our algorithm samples pairs of tuples. This enables the use of statistical inference techniques to estimate the distribution of $p(\varphi)$ and determine DC validity without evaluating all tuple pairs. As a result, the computational complexity of DC discovery becomes independent of the dataset size and depends solely on the chosen approximation factor.
- **Column Scalability**: Our algorithm explores a significantly smaller space of candidate DCs by avoiding the inclusion of independent predicates during the search of sound DCs. This improves efficiency on datasets with many attributes.
- **Dynamic Data**: Our algorithm updates the estimated distribution of $p(\varphi)$ as the data changes, allowing modifications in the dataset to directly propagate to the validity of the DCs.

LIMA is subdivided into three main components, illustrated in Figure 1: a lattice data structure, a scheduling engine and a tuple-pair sampling engine. We detail each component over this section, but in summary, the lattice is the main data structure of the algorithm, which stores, for each considered set of predicates φ, the distribution of $p(\varphi)$ as $Beta(k+1, n-k+1)$. The scheduling engine is a component that determines which nodes $\varphi_1, \varphi_2, \dots, \varphi_n$ of a given lattice state should have their distributions refined, to maximize the expected increase in information of the lattice in static discovery, or to guarantee uniformity of the sample in incremental discovery. Finally, the sampling engine is the component that selects the optimal strategy to evaluate each φ_i over the required number of tuple pairs, given a set of nodes scheduled for updates and the current lattice state.

The algorithm operates by iteratively alternating between scheduling and sampling stages, progressively refining our knowledge about each predicate set φ in the lattice.

4.1 Lattice

Let $P = P_1, P_2, \dots, P_k$ be the predicate space, and let $\Omega = 2^P$ denote the set of candidate DCs. This set is partially ordered by the $\subset$ operator, forming a lattice whose Hasse diagram is sketched in Figure 2. For each predicate set $\varphi \in \Omega$, we record the number $K(\varphi)$ of tuple pairs satisfying φ within a random sample of $N(\varphi)$ tuple pairs. As described in Section 3, the values of $K(\varphi)$ and $N(\varphi)$ are sufficient to model our knowledge of the joint probability of φ as $p(\varphi) \sim Beta(K(\varphi)+1, N(\varphi)-K(\varphi)+1)$ (See equation 1). This probabilistic characterization of $p(\varphi)$ is central to our approach, as it enables efficient decisions about properties of a DC φ and provides a formal method to quantify the certainty of those decisions.

Let $N : \Omega \to \mathbb{Z}^+$ and $K : \Omega \to \mathbb{Z}^+$ be mappings that provide $N(\varphi)$ and $K(\varphi)$ for each $\varphi \in \Omega$. We denote the pair $L = (N, K)$ as a lattice state. For any lattice state, we can determine which DCs $\varphi \in \Omega$ are satisfied and sound with confidence α, using the tests described in Sections 3.2 and 3.3.

Additionally, for any $\varphi \in \Omega$, we define an information measure $I(\varphi)$ equal to the Fischer information of the sample size $N(\varphi)$ in the distribution of $p(\varphi)$. This measure reflects how informative the distribution is: it takes low values when the true value of $ln(p(\varphi))$ is uncertain, and high values when the distribution of $ln(p(\varphi))$ is concentrated. We define the information of the entire lattice as the sum of the information of all predicate sets, $I(L) = \sum_{\varphi \in \Omega} I(\varphi)$.

4.2 Scheduling Engine

The goal of the scheduling engine is to determine how to update a lattice state L to obtain a new state L' with higher information: $I(L') > I(L)$. Formally, it takes as input a lattice state $L = (N, K)$ and a dataset $D = \{t_1, t_2, \dots\}$, and produces a schedule $S = (N', T)$ with $N' : \Omega \to \mathbb{Z}$ and $T \subseteq D \times D$.

The schedule maps each candidate DC φ to $N'(\varphi)$, indicating that the sampling engine must evaluate φ on $N'(\varphi)$ randomly drawn tuple pairs from T.

The behavior of this component differs depending on whether static or incremental DC discovery is being performed.

Static DC discovery. In static DC discovery, the lattice $L = (N, K)$ starts empty, with $N(\varphi) = 0$ for all $\varphi \in \Omega$, and is iteratively refined. To determine which predicate sets φ should have their distribution $p(\varphi)$ updated, the scheduling engine performs a two–stage process: First, the current lattice state is used to infer which predicate sets should be scheduled for refinement. To minimize wasted computation, the properties of DC validity presented in Section 2.2 restrict the search to predicate sets for which there is evidence in the current lattice that they may be informative:

- **Minimal:** A predicate set φ is considered for updates only if all its subsets show signs of not forming satisfied DCs in the current lattice state, i.e., $\frac{K(\varphi')}{N(\varphi')} > \epsilon \quad \forall \varphi' \subset \varphi$. [1]
- **Non-trivial:** Predicate sets containing logical implications are excluded from updates, since they can only yield trivial DCs.
- **Sound:** As discussed in Section 3, a DC is sound only if the marginal distributions of its predicates are conditioned on the others. To avoid exploring sets of independent predicates, a predicate set φ is considered for updates only if at least one of its subsets is already known to be sound.

Second, each predicate set selected for updates φ is assigned a number of tuple pairs $N'(\varphi)$ to be drawn from $T = \{(t_x, t_y) | (t_x, t_y) \in D \times D, x \neq y\}$ by the sampling engine. This will be added to the current sample of $N(\varphi)$ to produce a more informative lattice state. To ensure computation is allocated optimally, $N'(\varphi)$ is set to be proportional to the gradient of the information gain with respect to $N(\varphi)$. Figure 3 shows the schedule that would be generated for the lattice state displayed in Figure 2, with the left grid showcasing how the tuple pairs will be sampled uniformly.

Any lattice state enables the identification of predicate sets whose sampling would increase the overall information, implying that the refinement process could, in principle, continue indefinitely. To ensure termination, the scheduler ignores updates for nodes whose information gradient falls below a threshold $\Delta I_{\min}$. This threshold is derived from the information measure I, set to the largest value that guarantees exclusive sets of predicates φ for which $p(\varphi) = 0$ are proven satisfied once the gradient of $I(\varphi)$ with respect to $N(\varphi)$ drops below this $\Delta I_{\min}$. When the scheduler finds no predicate set eligible for update, the algorithm terminates, as all considered satisfied predicate sets have already been found.

Dynamic DC discovery. In dynamic DC discovery, we assume the existence of a lattice state $L = (N, K)$ obtained from static discovery with a subset of the data $D' \subset D$, and aim to construct a new lattice state L' that is statistically representative for D. To do so, a schedule is created so that all $N(\varphi)$ increase by the same proportion, sampling tuple pairs from the set $T = \{(t_x, t_y) | (t_x, t_y) \in D \times D \setminus D' \times D', x \neq y\}$. The right grid of Figure 3 illustrates how newly sampled tuple pairs are selected so that the overall sample becomes uniformly distributed.

[1] In practice, satisfaction tests are performed using the statistical framework presented in Section 3, as it is more robust.

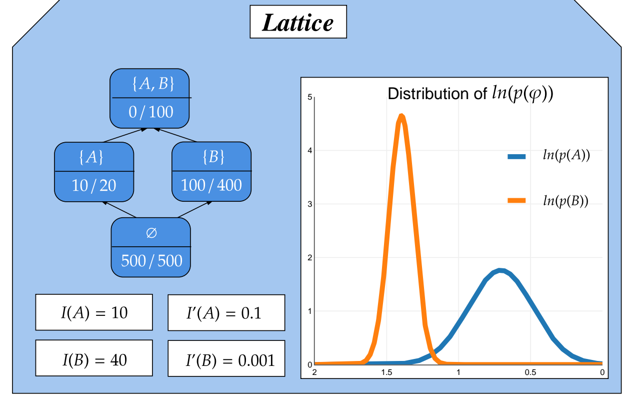


Figure 2: Lattice state $L = (N, K)$ for predicate space $P = \{A, B\}$, with visualizations of the distributions of $\ln(p(\{A\}))$ and $\ln(p(\{B\}))$ estimated with the available N and K. Their information measures I are also shown, with their derivatives with respect to the sample size I', illustrating how informative these distributions are, and how much additional information is expected from increasing the sample.

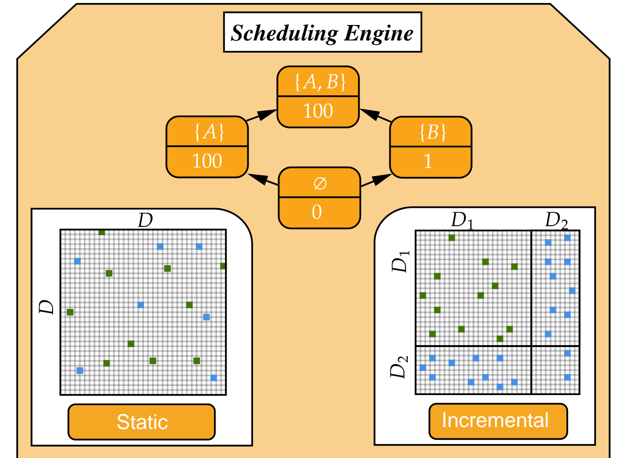


Figure 3: Schedule detailing the number of tuple pairs to be evaluated $N'(\varphi)$ for each φ in Figure 2 so as to maximize the expected information gain, and in blue the tuple pairs that will be used to sample the schedule. In static discovery, these tuple pairs are sampled from the full set of available tuple pairs and added to the current sample, in green. In incremental discovery, they are sampled from the newly introduced tuple pairs to ensure that the distributions remain statistically representative after the dataset grows.

4.3 Sampling Engine

The goal of the sampling engine is to determine the optimal way to sample a schedule. Formally, given a schedule $S = (N', T)$, the sampling engine produces a lattice state $L = (N', K')$, where $K'(\varphi)$ is the number of tuple pairs among the $N'(\varphi)$ drawn from T that satisfy all predicates in φ. Figure 4 shows the result of sampling the schedule illustrated in Figure 3, and provides visualizations of the different algorithms we propose for executing a schedule. While these algorithms are developed as part of our framework for discovering DCs, the contents of this section are relevant to all DC discovery methods for several reasons:

- We provide experimental evidence (Observation 4.1) demonstrating that a key assumption shared by all existing DC discovery algorithms fails to hold for large datasets, making most methods unsuitable for discovering DCs on datasets with more than twenty attributes.
- We introduce an alternative way to leverage the evidence set to compute $K'(\varphi)$, which is orders of magnitude faster than the state of the art for certain predicate spaces.

Initially, we assume:

$$N'(\varphi) = \begin{cases} N'(S) & \text{if } \varphi \in \Omega' \\ 0 & \text{otherwise} \end{cases}$$

meaning that the number of tuple pairs to be sampled, $N'(\varphi)$, is uniform across all nodes scheduled for update, $\varphi \in \Omega' \subset \Omega$, with a fixed value $N'(S)$. Section 4.3.1 details how to uniform schedules are sampled, and Section 4.3.2 shows how to sample any schedule by reducing it to uniform schedule sampling.

4.3.1 Sampling uniform schedules. We now present several methods for sampling uniform schedules, highlighting their strengths and weaknesses.

Node sampling. A straightforward way to sample $S = (N'(S), T)$ is shown in Algorithm 1. It begins by drawing $N'(S)$ tuple pairs uniformly from T, forming a subset $T' \subseteq T$ with $|T'| = N'(S)$. For each sampled pair $(t_x, t_y) \in T'$, we then evaluate all predicates $P_i \in \varphi$ on that tuple pair: $P_i(t_x, t_y)$. $K'(\varphi)$ is set to the number of tuple pairs out of the $N'(\varphi)$ that fulfill all predicates in φ. The temporal complexity of performing node sampling of Ω' on T' is $O(|T'| \sum_{\varphi \in \Omega'} |\varphi|)$.

Algorithm 1 Node Sampling

Input:
$\Omega' \leftarrow \{\varphi_i, \varphi_j, \dots\}$ Candidate DCs to sample
$T' \leftarrow$ Sample of $N'(S)$ tuple pairs (t_x, t_y)

Output:
$K' \leftarrow$ Number of tuple pairs in T' that satisfy each $\varphi \in \Omega'$

1: **procedure** NodeSampling(φ, T')
2: $\quad T_\varphi \leftarrow T'$
3: $\quad$ **for** $P_i \in \varphi$ **do**
4: $\quad\quad T_\varphi \leftarrow \{(t_x, t_y) | (t_x, t_y) \in T_\varphi, P_i(t_x, t_y)\}$
5: $\quad K'(\varphi) \leftarrow |T_\varphi|$
6: **for** $\varphi \in \Omega'$ **do** NodeSampling(φ, T')

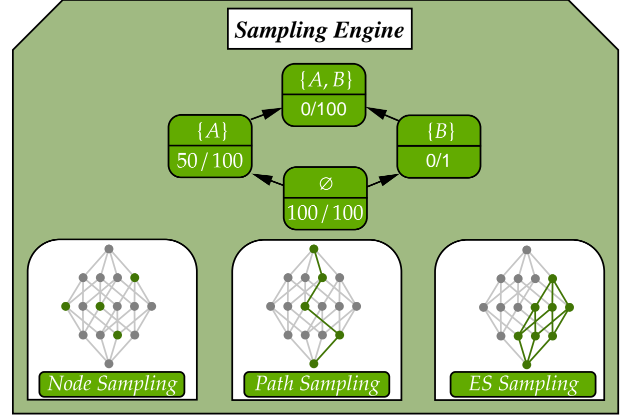


Figure 4: Result of sampling the schedule of Figure 3, showing the number of tuple pairs $K'(\varphi)$ that fulfill φ out of the uniformly drawn $N'(\varphi)$, and visualizations of different sampling algorithms.

Path sampling. Node sampling can be trivially optimized by interpreting the evaluation of φ on T' as tracing a path in the lattice from {} to φ, as shown in Algorithm 2. At each step, the set T' is refined by keeping only tuple pairs that fulfill the added predicate. Assuming the path taken is $\Omega' = (\{\}, \{P_{i_1}\}, \{P_{i_1}, P_{i_2}\}, \dots, \varphi)$, the cost of sampling φ is reduced to $O(\sum_{\varphi \in \Omega'} K'(\varphi))$. For any φ, there exist $|\varphi|!$ possible paths that may be traced to reach φ in the lattice. By using the current lattice state $L = (N, K)$, we can estimate the path that will minimize the cost by starting at the root and at each step moving towards the adjacent node φ' with smallest $\frac{K(\varphi')}{N(\varphi')}$.

If $|\Omega'| > 1$, sampling nodes independently may lead to the recomputation of partial results shared by nodes with overlapping paths. This can be avoided by traversing Ω' in a depth-first search, passing through each $\varphi \in \Omega'$ exactly once. Let $Parent : \Omega' \rightarrow \Omega'$ map each node φ to its parent $Parent(\varphi)$ in a tree spanning Ω'. In our case, given the current lattice state $L = (N, K)$, $Parent(\varphi) = \varphi'$ is defined as the $\varphi' \subset \varphi$ with smallest $\frac{K(\varphi)}{N(\varphi)}$. This tree defines a unique path from the root to each φ that may be traversed in a depth first search to compute $K'(\varphi)$ for all $\varphi \in \Omega'$ with minimal cost. The temporal complexity of performing path sampling of Ω' on T' is $O(\sum_{\varphi \in \Omega'} |K'(Parent(\varphi))|)$, with a required buffer of $O(|T'|)$ to maintain the DFS stack.

Evidence set sampling. As discussed in Section 2.4, previous research highlighted the redundancy of operating on individual tuple pairs [7]. This motivated the use of the evidence set. An evidence set $E : \Omega' \rightarrow 2^{T'}$ maps each set of predicates $\varphi \in \Omega'$ to the set of tuple pairs that satisfy exactly those predicates, denoted $E(\varphi)$. Algorithm 3.1, from [7], shows how to generate E from a predicate space P and a set of tuple pairs T', with cost $O(|P| \cdot |T'|)$.[2] This data structure provides an alternative way to compute $K'(\varphi)$. Instead of filtering tuple pairs one by one, it processes all tuple pairs that satisfy the same predicates together. Building on the approach in [7], Algorithm 3.2 presents a version of Algorithm 2

[2]In practice, it is enough to know the number of tuple pairs that satisfy a given set of predicates, i.e., $|E(\varphi)|$.

Algorithm 2 Path Sampling

Input:

$Parent \leftarrow$ Parent of each $\varphi \in \Omega'$ forming a tree in Ω'
$T' \leftarrow$ Sample of $N'(S)$ tuple pairs (t_x, t_y)

Output:

$K' \leftarrow$ Number of tuple pairs in T' that satisfy each $\varphi \in \Omega'$

```
1: procedure PathSampling(φ, T_φ, Parent)
2:     K'(φ) ← |T_φ|
3:     for P_i ∈ {P_i | Parent(φ ∪ {P_i}) = φ} do
4:         φ_next ← φ ∪ {P_i}
5:         T_{φ_next} ← {(t_x, t_y) | (t_x, t_y) ∈ T_φ, P_i(t_x, t_y)}
6:         PathSampling(φ_next, T_{φ_next}, Parent)
7: PathSampling({}, T', Parent)
```

that filters the evidence set rather than the set of tuple pairs. The temporal complexity of performing evidence set sampling of Ω' on T' is $O(\sum_{\varphi \in \Omega', |E(\varphi)|>0} 2^{|\varphi|})$, plus the overhead from computing the evidence set $O(|P| \cdot |T'|)$.

Algorithm 3.1 Evidence Set Building

Input:

$P \leftarrow$ Predicate space $\{P_1, P_2, \dots\}$
$T' \leftarrow$ Sample of $N'(S)$ tuple pairs (t_x, t_y)

Output:

$E(\varphi) \leftarrow$ Tuple pairs in T' fulfilling exactly $\varphi \subseteq P$

```
1: procedure BuildEvidenceSet(P, T')
2:     E ← Map with default empty value.
3:     for (t_x, t_y) ∈ T' do
4:         φ ← {}
5:         for P_i ∈ P do
6:             if P_i(t_x, t_y) then
7:                 φ ← φ ∪ {P_i}
8:         E(φ) ← E(φ) ∪ {(t_x, t_y)}
9: BuildEvidenceSet(P, T')
```

We denote the size of the evidence set $|E|$ as the number of $\varphi \in \Omega'$ with nonzero $|E(\varphi)|$. Prior work [5, 7, 15] assumes that the evidence set is typically much smaller than the set of tuple pairs $|T'|$ from which it is built. Consequently, computing $K'(\varphi)$ with Algorithm 3.2 should be orders of magnitude faster than with Algorithm 2. The assumption $|E| \ll |T'|$ underlies all DC discovery algorithms that rely on the evidence set [5, 7, 9, 14–16, 19]. However, for large databases with numerous predicates, the combinatorial explosion of predicate signatures ensures that the evidence set is dense, invalidating the assumption $|E| \ll |T'|$. Figure 5 presents experimental evidence showcasing that this assumption breaks down for large datasets, as the upper bound for $|E|$ grows exponentially with $|P|$, easily surpassing $|T'|$. This aligns with the experimental findings of [5, 7, 15, 16, 19], which report that the cost of DC enumeration over evidence sets grows exponentially with the number of predicates. We summarize this behavior in the following observation:

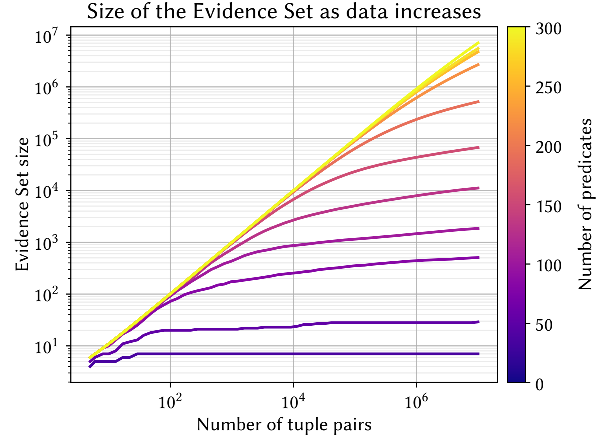


Figure 5: Size of the evidence set $|E|$ evolution as the number of tuple pairs $|T'|$ from the Flights dataset used to generate it increases $E = \text{BuildEvidenceSet}(P, T')$, for multiple sizes of the predicate space $|P|$. The trend showcases how the assumption that $|E| \ll |T'|$ only holds true for small predicate spaces, which cannot be guaranteed for large databases.

Observation 4.1. *An evidence set E built for a predicate space P on a large dataset will be dense: $|E| = O(2^{|P|})$.*

This observation has profound implications to all those DC discovery algorithms that rely on the assumption of sparse evidence sets. In the context of our algorithm, it implies that an evidence set built from a reduced sample of tuple pairs is not bound in size, and may therefore be as large as the sample itself. We summarize the consequence of this fact with for evidence set sampling in the following observation:

Observation 4.2. *For large predicate spaces, evidence-set-based sampling degenerates to path sampling while incurring the additional overhead of constructing and storing the evidence set.*

Algorithm 3.2 Evidence Set Sampling

Input:

$\Omega' \leftarrow \{\varphi_i, \varphi_j, \dots\}$ Candidate DCs to sample
$Parent \leftarrow$ Parent of each $\varphi \in \Omega'$ forming a tree in Ω'
$T' \leftarrow$ Sample of $N'(S)$ tuple pairs (t_x, t_y)

Output:

$K' \leftarrow$ Number of tuple pairs in T' that satisfy each $\varphi \in \Omega'$

```
1: procedure EvidenceSetSampling(φ, Ω_φ, E, Parent)
2:     K'(φ) ← Σ_{φ'∈Ω_φ} |E(φ')|
3:     for P_i ∈ {P_i | Parent(φ ∪ {P_i}) = φ} do
4:         φ_next ← φ ∪ {P_i}
5:         Ω_{φ_next} ← {φ' | φ' ∈ Ω_φ, P_i ∈ φ}
6:         EvidenceSetSampling(φ_next, Ω_{φ_next}, E, Parent)
7: P ← ⋃_{φ∈Ω'} φ
8: E ← BuildEvidenceSet(P, T')
9: EvidenceSetSampling({}, Ω', E, Parent)
```

Dense evidence set sampling. We present a dynamic programming algorithm that significantly reduces redundant computations for dense evidence sets. The key contribution is that our algorithm computes $K'(\varphi)$ from a dense evidence set faster than evidence set sampling, whilst attaining no additional memory footprint. The algorithm is based on the zeta transform for subset sums of functions from [6]. For any set of predicates φ, the target $K'(\varphi)$ can be expressed as a sum of elements of the evidence set $K'(\varphi) = \sum_{\varphi' \supseteq \varphi} E(\varphi')$. Informally, the number of tuple pairs that satisfy a set of predicates $K'(\varphi)$, equals the number of tuple pairs that exactly fulfill φ, $E(\varphi)$, plus the ones that exactly fulfill φ and any other predicates: $E(\varphi \cup \{P_i, P_j, \ldots\})$.

We denote as $E(\varphi, P')$ the set of tuple pairs that exactly fulfill φ plus any subset of $P' \subseteq P$. If $P' = \{\}$, $E(\varphi, P')$ is the evidence set $E(\varphi)$, and if $P' = P$, $|E(\varphi, P')|$ equals $K'(\varphi)$. Additionally, the following recursive rule can be established:

$$E(\varphi, P' + P_i) \begin{cases} E(\varphi, P') & \text{if } P_i \in \varphi \\ E(\varphi, P') \cup E(\varphi + P_i, P') & \text{otherwise} \end{cases}$$

Algorithm 4 implements a bottom up dynamic programming approach to use this recurrence to compute each $K'(\varphi)$. Table 2 illustrates the full execution of this algorithm for a predicate space $P = \{A, B, C\}$. It shows that, at the end of each iteration, $|E(\varphi, P')|$ exactly equals the number of tuple pairs that satisfy precisely the predicates in φ together with any predicates in P'. The time complexity of performing dense evidence set sampling over Ω' on T' is $O(\sum_{\varphi \in \Omega', |E(\varphi)|>0} |\varphi|)$, plus the overhead of constructing the evidence set $O(|P| \cdot |T'|)$, and an additional $O(|\Omega'|)$ buffer to store intermediate results.

Algorithm 4 Dense Evidence Set Sampling

Input:
$\Omega' \leftarrow \{\varphi_i, \varphi_j, \ldots\}$ Candidate DCs to sample
$T' \leftarrow$ Sample of $N'(S)$ tuple pairs (t_x, t_y)

Output:
$K' \leftarrow$ Number of tuple pairs in T' that satisfy each $\varphi \in \Omega'$

```
1: P ← ⋃_{φ∈Ω'} φ
2: E ← BuildEvidenceSet(P, T')
3: for φ' ∈ Ω' do
4:     K'(φ) ← E(φ)
5: for P_i ∈ P do
6:     for φ' ∈ Ω' do
7:         if P_i ∉ φ then
8:             K'(φ) ← K'(φ) + K'(φ + P_i)
```

4.3.2 Sampling general schedules. Table 3 summarizes the temporal and spatial complexities of the sampling algorithms described previously for uniform schedules. For any uniform schedule to be sampled $S = (N'(S), T)$, the sampling engine estimates the cost of each method with the information available in the current lattice state $L = (N, K)$. The method with lowest estimated temporal cost that also fits into memory is chosen as the optimal sampling strategy for this uniform schedule.

φ	$\lvert E(\varphi, \{\})\rvert$	$\lvert E(\varphi, \{A\})\rvert$	$\lvert E(\varphi, \{A, B\})\rvert$	$\lvert E(\varphi, P)\rvert$
$\{\}$	0	0+4=4	4+4=8	8+6=14
$\{A\}$	4	4	4+1=5	5+3=8
$\{B\}$	3	3+1=4	4	4+2=6
$\{C\}$	2	2+2=4	4+2=6	6
$\{A, B\}$	1	1	1	1+1=2
$\{A, C\}$	2	2	2+1=3	3
$\{B, C\}$	1	1+1=2	2	2
$\{A, B, C\}$	1	1	1	1

Table 2: Iterations of Dense ES sampling over a predicate space $P = \{A, B, C\}$ assuming 4 tuples only satisfy A: $E(\{A\}) = 4$, 3 tuples only satisfy B: $E(\{B\}) = 3$, only 1 tuple satisfies both A and B but not C: $E(\{A, B\}) = 3$, and so on.

Method	Time	Memory
Node	$O(\lvert T'\rvert \sum_{\varphi \in \Omega'} \lvert\varphi\rvert)$	$O(1)$
Path	$O(\sum_{\varphi \in \Omega'} \lvert K'(Parent(\varphi))\rvert)$	$O(\lvert T'\rvert)$
ES	$O(\sum_{\varphi \in \Omega', \lvert E(\varphi)\rvert>0} 2^{\lvert\varphi\rvert}) + O(\lvert P\rvert \cdot \lvert T'\rvert)$	$O(\lvert E\rvert \cdot \lvert P\rvert)$
Dense ES	$O(\sum_{\varphi \in \Omega', \lvert E(\varphi)\rvert>0} \lvert\varphi\rvert) + O(\lvert P\rvert \cdot \lvert T'\rvert)$	$O(\lvert\Omega'\rvert)$

Table 3: Computational complexities of uniform schedule sampling strategies.

Finally, sampling a schedule $S = (N', T)$ can be reduced to sampling a collection of uniform schedules. This is done by partitioning the predicate sets to be sampled:

$$\Omega' = \{\varphi \mid N'(\varphi) > 0\},$$

into buckets, where Each subset Ω'_i is then processed as an independent uniform schedule $S_i = (2^i, T)$, as:

$$\Omega'_i = \{\varphi \mid N'(\varphi) \in [2^i, 2^{i+1})\}.$$

In practice, this may cause some φ to be evaluated on more tuple pairs than originally scheduled by up to a factor of two. This only results in a more informative lattice state than expected, which is never an undesirable byproduct.

After the schedule has been sampled, yielding $L' = (N', K')$, the newly obtained information is incorporated into the current lattice state $L_i = (N_i, K_i)$ to produce the updated lattice state $L_{i+1} = (N_{i+1}, K_{i+1})$, as:

$$N_{i+1}(\varphi) = N_i(\varphi) + N'(\varphi)$$
$$K_{i+1}(\varphi) = K_i(\varphi) + K'(\varphi).$$

As discussed in Section 3, for our statistical method to be valid and unbiased, $K_i(\varphi)$ and $N_i(\varphi)$ must always correspond to the evaluation of φ on a uniformly drawn sample of tuple pairs. By induction, we assume that the current lattice state is based on uniformly drawn samples. Since the additional values $K'(\varphi)$ and $N'(\varphi)$ are also obtained from uniformly drawn samples, the updated quantities $K_{i+1}(\varphi)$ and $N_{i+1}(\varphi)$ likewise correspond to a uniformly drawn sample. Thus, LIMA guarantees that, at any point in time, its estimated distributions are unbiased, with their information monotonically increasing as more uniform samples are introduced.

5 EXPERIMENTAL RESULTS

In this section, we evaluate the performance of our algorithm and compare it with state-of-the-art methods [9, 15–19][3]. We evaluate the scalability of static DC discovery algorithms by performing DC discovery on a large dataset while progressively increasing its dimensions, following the experimental methodology of [5, 7, 15, 16, 19]. For this experiment, we use the largest dataset commonly employed in the DC discovery literature[4]. The quality of the discovered DCs is evaluated by comparing them against manually curated sets of golden DCs for several datasets used in the DC discovery literature, as done in [4, 7, 10, 13, 15]. Finally, we assess the ability of our algorithm to adapt to incremental data by measuring the time required to update the discovered DCs after adding new data slices of varying sizes, also for several datasets used in the DC literature, following the methodology of [17, 18].

5.1 Scalability

As described in Section 2.3, existing DC discovery algorithms display sharp increases in computational cost as datasets grow large. Experimental results in the literature show that the cost of DC discovery scales quadratically with the number of tuples due to the construction of the evidence set [5, 7, 15, 16]. Additional experiments demonstrate that the cost also grows exponentially with the number of attributes, due to the large size of the candidate DC space [5, 7, 15, 16, 19]. Together, these factors render DC discovery infeasible on large datasets.

To experimentally evaluate the scalability of DC discovery algorithms, we follow the standard approach in the literature and run each algorithm while progressively increasing the number of rows or columns provided as input. The only dataset used in prior work that enables scalability analysis on large datasets is the Flights dataset, which, after aggregating data from multiple time periods, contains over ten million tuples and more than 60 attributes. While some of the DC discovery algorithms evaluated allow for parallelization using appropriate hardware, our goal is to measure the raw computation each algorithm requires to perform DC discovery. To this end, experiments in this section are run on a single core, ensuring execution times directly correlate to the computational complexity of each algorithm.

First, we evaluate row scalability by discovering DCs on slices of the Flights dataset with an increasing number of tuples, while fixing the set of attributes to those used in prior work on the Flights dataset [9, 15, 16]. Figure 6a summarizes the execution times for DC discovery. The results are consistent with the literature and show that existing algorithms scale quadratically with the number of tuples in the data [5, 7, 15, 16]. In contrast, the computational cost of LIMA depends solely on the approximation factor ϵ. Figure 6b reports the corresponding memory footprints. These results also align with prior work, indicating that when the number of attributes is small, the size of the evidence set remains bounded. Since LIMA does not build the full evidence set, it maintains lower memory footprint even for smaller datasets.

[3] All experiments were conducted on an Intel Xeon E5-2660 v3 (10 cores, 20 threads and 2.6GHz clock) running Ubuntu-based GNU-Linux (24.04.1 LTS) and 16GB of DRAM.
[4] *Carrier on-time performance* dataset, available at https://www.transtats.bts.gov, and commonly referred to as the Flights dataset in the DC discovery literature.

Next, we evaluate column scalability by discovering DCs on slices of the Flights dataset with an increasing number of attributes, while fixing the number of tuples to 5000. Figure 6c summarizes the execution times for DC discovery. The results are consistent with the literature and show that existing algorithms scale exponentially with the number of attributes [5, 7, 15, 16, 19]. In contrast, the computational cost of LIMA is much less affected by this increase, as it does not require exploring the full space of predicate combinations. Figure 6d reports the corresponding memory footprints. These results also align with our findings in Figure 5, indicating that as the number of attributes grows, the evidence set that must be stored in state-of-the-art algorithms increases exponentially in size, making them impractical. By contrast, LIMA avoids storing the full evidence set, making it well suited for large predicate spaces.

5.2 Quality

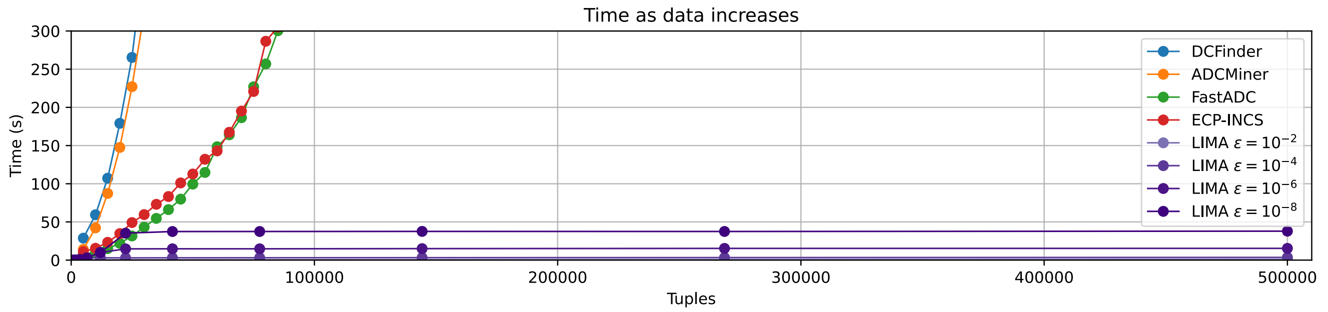
As shown before, LIMA has significantly lower computational requirements to perform DC discovery. However, this is achieved by limiting the analyzed data and the space of candidate DCs explored. Hence, it is imperative to ensure that the quality of the DCs discovered by LIMA is not significantly degraded by these optimizations. In order to assess the quality of the discovered DCs, we follow the literature in comparing the discovered DCs to a set of manually generated golden DCs, which are representative of the overall DCs of the data [7, 9, 10, 15]. We use the publicly available set of golden DCs from [10] to measure the precision and recall.

Table 4 summarizes the precision of the results, defined as the proportion of discovered DCs that belong to the set of golden DCs. The results for traditional DC discovery algorithms are consistent with the literature, showing that these algorithms typically discover many more DCs than the true number of constraints in the data, which results in low precision [7, 10, 15]. This behavior has been attributed to the agglomeration of independent predicates by DC discovery algorithms [10]. In contrast, LIMA relies on a more restrictive definition of DC validity that requires soundness, meaning that DCs must represent some statistically significant relationship among its predicates. This results in significantly higher precisions due to rejecting any DC constructed using independent predicates.

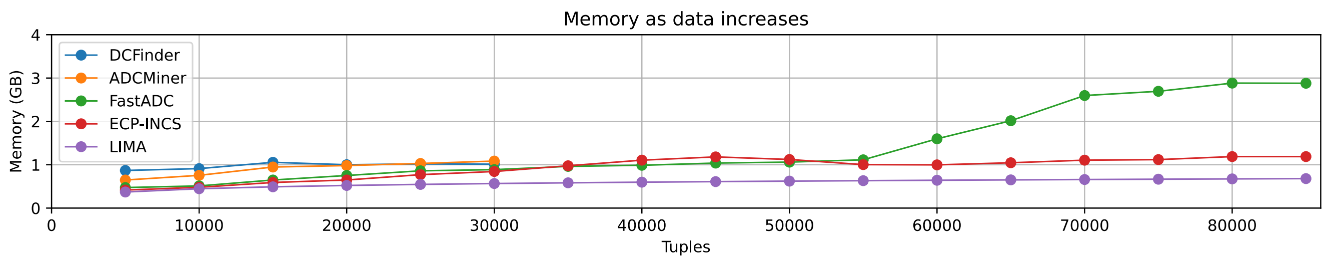
Table 5 summarizes the recall of the results, defined as the proportion of golden DCs contained in the set of discovered DCs. These results are also consistent with the literature, as the large number of DCs produced by traditional approaches makes it likely for golden DCs to appear among the discovered ones, resulting in high recall values [7, 10, 15]. In contrast, LIMA substantially reduces the space of candidate DCs that is explored, and most importantly, the number of discovered DCs. Despite this restriction, the observed recall remains comparable to that of existing algorithms. This is an encouraging result, as it shows that our soundness-based exploration strategy significantly lowers the computational cost of DC discovery without missing a substantial number of valid DCs.

5.3 Incremental DC Discovery

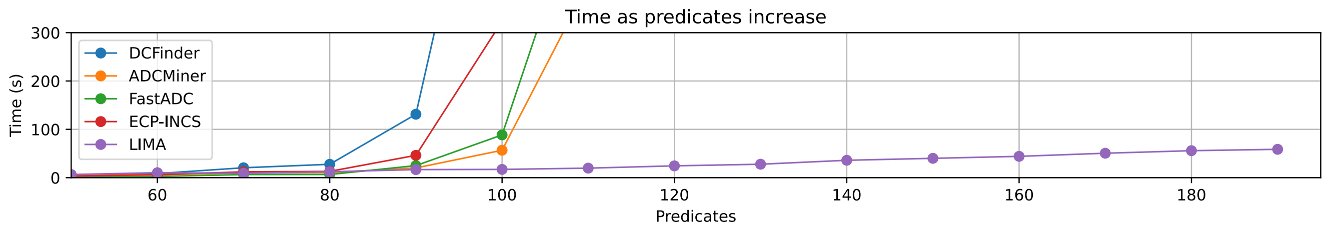
While LIMA is primarily designed for static DC discovery, its novel approach to computing and representing $p(\varphi)$ enables efficient updates as the data evolves. We evaluate this capability by comparing our algorithm for updating the set of DCs valid on a dataset D,



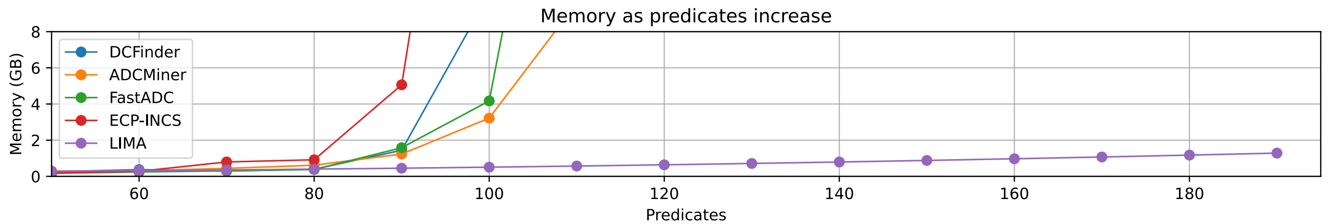
(a) Execution times for DC discovery on the Flights dataset with 20 attributes and increasing rows. Approximation factors with negligible differences are grouped for clarity. The results show that the cost of DC discovery grows quadratically with the number of tuples in the data, whereas the complexity of our algorithm depends solely on the desired approximation factor ϵ, enabling DC discovery on datasets with a large number of rows.



(b) Memory footprints for DC discovery on the Flights dataset with 20 attributes and increasing rows. Approximation factors with negligible differences are grouped for clarity. As shown in Figure 5, when the number of attributes (and thus the number of predicates) is small, the size of the evidence set remains bounded and small enough to fit into memory.



(c) Execution times for DC discovery on the Flights dataset with 5000 rows and an increasing number of attributes (and thus predicates). Approximation factors with negligible differences are grouped for clarity. The results show that the cost of DC discovery grows exponentially with the number of attributes, due to the need to explore most predicate combinations. In contrast, our algorithm avoids this expensive exploration by leveraging the soundness property, substantially reducing the computational cost of DC discovery.



(d) Memory footprints for DC discovery on the Flights dataset with 5000 rows and an increasing number of attributes (and thus predicates). Approximation factors with negligible differences are grouped for clarity. As shown in Figure 5, when the number of attributes grows, the size of the evidence set is unbounded and grows too large to be stored.

Figure 6: Experimental results demonstrating the improved scalability of our algorithm compared to the state of the art, in terms of both row and column time and memory scalability.

Algorithm		ϵ	Dataset					
			Airport	Flights	Food	Hospital	Ncvoter	Tax
Algorithm	DCFinder	10^{-8}	0.011	0.0081	0.077	0.21	0.043	0.008
		10^{-6}	0.026	0.0074	0.068	0.18	0.037	0.007
		10^{-4}	0.020	0.0070	0.072	0.16	0.031	0.003
		10^{-2}	0.035	0.0049	0.080	0.04	0.051	0.009
	ADCMiner	10^{-8}	0.009	0.0011	0.076	0.21	0.041	0.002
		10^{-6}	0.024	0.0002	0.071	0.16	0.027	0.001
		10^{-4}	0.013	0.0003	0.077	0.16	0.030	0.002
		10^{-2}	0.018	0.0006	0.078	0.03	0.044	0.011
	FastADC	10^{-8}	0.009	0.0011	0.076	0.21	0.041	0.002
		10^{-6}	0.024	0.0002	0.071	0.16	0.027	0.001
		10^{-4}	0.013	0.0003	0.077	0.16	0.030	0.002
		10^{-2}	0.018	0.0006	0.078	0.03	0.044	0.011
	ECP-INCS	10^{-8}	0.012	0.0081	0.076	0.21	0.040	0.008
		10^{-6}	0.021	0.0069	0.072	0.18	0.029	0.008
		10^{-4}	0.020	0.0069	0.074	0.17	0.031	0.006
		10^{-2}	0.018	0.0047	0.080	0.04	0.055	0.011
	LIMA	10^{-8}	**0.80**	**0.96**	**0.93**	**0.92**	**0.85**	**0.82**
		10^{-6}	**0.66**	**0.89**	**0.78**	**0.86**	**0.85**	**0.76**
		10^{-4}	**0.60**	**0.76**	**0.66**	**0.82**	**0.74**	**0.61**
		10^{-2}	**0.25**	**0.73**	**0.60**	**0.43**	**0.61**	**0.22**

Table 4: Precision of the sets of discovered DCs, showing how the large number of false positives is mitigated by adopting a more restrictive DC validity. Best per dataset and ϵ in bold.

Algorithm		ϵ	Dataset					
			Airport	Flights	Food	Hospital	Ncvoter	Tax
Algorithm	DCFinder	10^{-8}	**1**	**0.984**	**0.875**	**1**	**1**	**1**
		10^{-6}	**1**	**0.968**	**0.75**	0.91	**1**	0.765
		10^{-4}	**1**	**0.927**	**0.5**	0.63	**0.9**	0.411
		10^{-2}	**0.5**	0.806	**0.375**	0.27	**0.9**	0.176
	ADCMiner	10^{-8}	**1**	**0.984**	**0.875**	**1**	**1**	**1**
		10^{-6}	**1**	0.946	**0.75**	**0.93**	**1**	**0.790**
		10^{-4}	**1**	0.901	**0.5**	**0.69**	**0.9**	0.406
		10^{-2}	**0.5**	0.806	**0.375**	**0.29**	**0.9**	**0.189**
	FastADC	10^{-8}	**1**	**0.984**	**0.875**	**1**	**1**	**1**
		10^{-6}	**1**	0.946	**0.75**	**0.93**	**1**	**0.790**
		10^{-4}	**1**	0.901	**0.5**	**0.69**	**0.9**	0.406
		10^{-2}	**0.5**	0.806	**0.375**	**0.29**	**0.9**	**0.189**
	ECP-INCS	10^{-8}	**1**	**0.984**	**0.875**	**1**	**1**	**1**
		10^{-6}	**1**	**0.968**	**0.75**	0.91	**1**	0.765
		10^{-4}	**1**	0.946	**0.5**	0.63	**0.9**	**0.442**
		10^{-2}	**0.5**	**0.823**	**0.375**	0.27	**0.9**	0.176
	LIMA	10^{-8}	**1**	0.976	**0.875**	**1**	**1**	**1**
		10^{-6}	**1**	0.937	**0.75**	**0.93**	**1**	0.778
		10^{-4}	**1**	0.901	**0.5**	**0.69**	**0.9**	0.428
		10^{-2}	**0.5**	0.812	**0.375**	0.27	**0.9**	0.176

Table 5: Recall of the sets of discovered DCs, showing how our optimizations do not lead to a noticeable increase in false negatives. Best per dataset and ϵ in bold.

Dataset	Properties		$\lvert\Delta D\rvert = 0.1\% \cdot \lvert D\rvert$			$\lvert\Delta D\rvert = 1\% \cdot \lvert D\rvert$			$\lvert\Delta D\rvert = 10\% \cdot \lvert D\rvert$			$\lvert\Delta D\rvert = 30\% \cdot \lvert D\rvert$		
Name	#Cols	#Rows	3DC	IncDC	LIMA	3DC	IncDC	LIMA	3DC	IncDC	LIMA	3DC	IncDC	LIMA
Airport	11	55K	0.71	28.50	**0.09**	0.80	84.56	**0.15**	**0.25**	786.17	0.81	**0.76**	–	1.66
Hospital	15	115K	**0.05**	23.48	0.06	0.11	24.35	**0.08**	0.59	51.29	**0.24**	1.94	–	**0.86**
NCVoter	15	675K	1.26	253.67	**0.15**	4.88	292.17	**0.62**	44.83	–	**4.25**	183.77	–	**11.61**
Tax	15	100K	0.13	40.57	**0.06**	0.42	123.82	**0.20**	2.57	–	**0.95**	7.97	–	**2.82**
Flight	17	500K	0.72	–	**0.14**	4.67	–	**0.55**	47.63	–	**5.09**	182.44	–	**14.31**

Table 6: Times (s) to update the set of valid DCs after increasing the data by ΔD, showing that our algorithm preserves the scalability benefits observed in Figure 6 when adapting the set of valid DCs. Best per dataset and ΔD in bold. Timeouts show -.

starting from the DCs valid on a subset $D - \Delta D$, against the only two existing incremental DC discovery algorithms [17, 18]. We compare them to LIMA executed with $\epsilon = 10^{-8}$, which has been shown to produce comparable results to exact DC discovery [10]. We compare running times of LIMA to the experimental results published in [17] obtained with comparable hardware.

Table 6 summarizes these results. It shows that LIMA's ability to update the set of valid DCs as the data grows is comparable to the state of the art 3DC on small datasets. For larger datasets, the results follow the same trend observed in Figure 6a, demonstrating that the reduced computational cost of LIMA on datasets with many rows also carries over to the incremental setting. This is because extending the sample used to infer $p(\varphi)$ is significantly less expensive than evaluating all newly formed pairs of tuples.

6 CONCLUSIONS

This paper presented the Lattice Information Maximization Algorithm (LIMA) for discovering approximate DCs on large databases. By leveraging statistical methods to model the distribution of the joint probabilities of predicate sets, LIMA can make decisions about DC validity from a reduced sample of tuple pairs. We experimentally demonstrated how this enables the discovery of approximate DCs with significantly lower computational requirements than the state of the art, greatly improving both row and column scalability. Furthermore, LIMA adopts a more restrictive definition of DC validity, which reduces the number of false positives by orders of magnitude while having a negligible impact on false negatives. Finally, although the algorithm is primarily designed for static DC discovery, its probabilistic modeling of joint predicate probabilities supports efficient updates as data evolves, making LIMA well suited for incremental DC discovery.

ACKNOWLEDGMENTS

This work is supported by the Horizon Europe Programme under GA.101135513 (CyclOps) and the Spanish Ministerio de Ciencia e Innovación under project PID2024-161707OB-I00 / AEI /10.13039/501100011033 (FeeD). Anna Queralt is a Serra-Húnter fellow. Eduardo Almeida is funded by the CNPQ grants 302909/2022-2 and 444192/2024-7. Albert Martin is funded by the predoctoral program AGAUR-FI grants (2025 FI-1 00967) Joan Oró, which is backed by the Department of Research and Universities of the Generalitat of Catalonia, as well as the European Social Plus Fund.

REFERENCES

[1] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. 1999. Consistent query answers in inconsistent databases. In *Proceedings of the Eighteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (Philadelphia, Pennsylvania, USA) *(PODS '99)*. Association for Computing Machinery, New York, NY, USA, 68–79. https://doi.org/10.1145/303976.303983

[2] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. 2003. Answer sets for consistent query answering in inconsistent databases. *Theory Pract. Log. Program.* 3, 4 (July 2003), 393–424. https://doi.org/10.1017/S1471068403001832

[3] Naser Ayat, Hamideh Afsarmanesh, Reza Akbarinia, and Patrick Valduriez. 2012. Pay-As-You-Go Data Integration Using Functional Dependencies. In *Multidisciplinary Research and Practice for Information Systems*. 375–389.

[4] 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 on Conference on Information and Knowledge Management (CIKM '16)*. Association for Computing Machinery, New York, NY, USA, 1803–1812. https://doi.org/10.1145/2983323.2983781

[5] Tobias Bleifuß, Sebastian Kruse, and Felix Naumann. 2017. Efficient denial constraint discovery with hydra. *Proc. VLDB Endow.* 11, 3 (Nov. 2017), 311–323. https://doi.org/10.14778/3157794.3157800 Number: 3.

[6] Charalambos A Charalambides. 2018. *Enumerative combinatorics.* Chapman and Hall/CRC.

[7] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. 2013. Discovering denial constraints. *Proc. VLDB Endow.* 6, 13 (2013), 1498–1509. https://doi.org/10.14778/2536258.2536262 Number: 13.

[8] Norman L Johnson, Samuel Kotz, and Narayanaswamy Balakrishnan. 1995. *Continuous univariate distributions, volume 2.* Vol. 2. John wiley & sons.

[9] Ester Livshits, Alireza Heidari, Ihab F Ilyas, and Benny Kimelfeld. [n.d.]. Approximate Denial Constraints. *Proceedings of the VLDB Endowment* 13, 10 ([n. d.]).

[10] Albert Martin, Eduardo C. De Almeida, Oscar Romero, and Anna Queralt. 2025. How and Why False Denial Constraints are Discovered. *Proceedings of the VLDB Endowment* 18, 10 (June 2025), 3477–3489. https://doi.org/10.14778/3748191.3748209

[11] Keisuke Murakami and Takeaki Uno. 2013. Efficient algorithms for dualizing large-scale hypergraphs. In *2013 Proceedings of the Fifteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM, 1–13.

[12] Thorsten Papenbrock and Felix Naumann. 2017. Data-driven Schema Normalization. In *Proceedings of the 20th International Conference on Extending Database Technology, EDBT 2017, Venice, Italy, March 21-24, 2017*, Volker Markl, Salvatore Orlando, Bernhard Mitschang, Periklis Andritsos, Kai-Uwe Sattler, and Sebastian Breß (Eds.). OpenProceedings.org, 342–353. https://doi.org/10.5441/002/EDBT.2017.31

[13] Marcel Parciak, Sebastiaan Weytjens, Niel Hens, Frank Neven, Liesbet M Peeters, and Stijn Vansummeren. 2025. Measuring approximate functional dependencies: a comparative study. *The VLDB Journal* 34, 4 (2025), 56.

[14] Eduardo H. M. Pena and Eduardo Cunha de Almeida. 2018. BFASTDC: A Bitwise Algorithm for Mining Denial Constraints. In *Database and Expert Systems Applications*, Sven Hartmann, Hui Ma, Abdelkader Hameurlain, Günther Pernul, and Roland R. Wagner (Eds.). Springer International Publishing, Cham, 53–68. https://doi.org/10.1007/978-3-319-98809-2_4

[15] Eduardo H. M. Pena, Eduardo C. de Almeida, and Felix Naumann. 2019. Discovery of approximate (and exact) denial constraints. *Proc. VLDB Endow.* 13, 3 (Nov. 2019), 266–278. https://doi.org/10.14778/3368289.3368293 Number: 3.

[16] Eduardo H. M. Pena, Fabio Porto, and Felix Naumann. 2022. Fast Algorithms for Denial Constraint Discovery. *Proc. VLDB Endow.* 16, 4 (2022), 684–696. https://doi.org/10.14778/3574245.3574254 Number: 4.

[17] Eduardo H. M. Pena, Fabio Porto, and Felix Naumann. 2024. Discovering Denial Constraints in Dynamic Datasets. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. 3546–3558. https://doi.org/10.1109/ICDE60146.2024.00273 ISSN: 2375-026X.

[18] Chaoqin Qian, Menglu Li, Zijing Tan, Ai Ran, and Shuai Ma. 2023. Incremental discovery of denial constraints. *The VLDB Journal* 32, 6 (Nov. 2023), 1289–1313. https://doi.org/10.1007/s00778-023-00788-y Number: 6.

[19] Renjie Xiao, Zijing Tan, Haojin Wang, and Shuai Ma. 2022. Fast approximate denial constraint discovery. *Proc. VLDB Endow.* 16, 2 (Oct. 2022), 269–281. https://doi.org/10.14778/3565816.3565828 Number: 2.

[20] Chen Ye, Haoyang Duan, Hua Zhang, Yifan Wu, and Guojun Dai. 2024. Learned Query Optimization by Constraint-Based Query Plan Augmentation. *Mathematics* 12, 19 (2024), 3102.