Incremental Rule Discovery in Response to Parameter Updates

HAOXIAN CHEN[*], ShanghaiTech University, China
WENFEI FAN, Shenzhen Institute of Computing Sciences, China, University of Edinburgh, United Kingdom, and Beihang University, China
JIAYE ZHENG, ShanghaiTech University, China

This paper studies incremental rule discovery. Given a dataset $\mathcal{D}$, rule discovery is to mine the set of all rules on $\mathcal{D}$ such that their supports and confidences are above thresholds σ and δ, respectively. We formulate incremental problems in response to updates $\Delta\sigma$ and/or $\Delta\delta$, to compute rules added and/or removed with respect to $\sigma + \Delta\sigma$ and $\delta + \Delta\delta$. The need for studying the problems is evident since practitioners often want to adjust their support and confidence thresholds during discovery. The objective is to minimize unnecessary recomputation during the adjustments, not to restart the costly discovery process from scratch. As a testbed, we consider entity enhancing rules, which subsume popular data quality rules as special cases. We develop three incremental algorithms in response to $\Delta\sigma$, $\Delta\delta$ and both. We show that relative to a batch discovery algorithm, these algorithms are bounded, *i.e.,* they incur the minimum cost among all incrementalizations of the batch one, and parallelly scalable, *i.e.,* they guarantee to reduce runtime when given more processors. Using real-life data, we empirically verify that the incremental algorithms outperform the batch counterpart by up to 658× when $\Delta\sigma$ and $\Delta\delta$ are either positive or negative.

CCS Concepts: • **Information systems** → **Information integration**.

Additional Key Words and Phrases: Rule discovery, incremental discovery

ACM Reference Format:
Haoxian Chen, Wenfei Fan, and Jiaye Zheng. 2025. Incremental Rule Discovery in Response to Parameter Updates. *Proc. ACM Manag. Data* 3, 3 (SIGMOD), Article 175 (June 2025), 30 pages. https://doi.org/10.1145/3725312

1 Introduction

Rule discovery has been studied for decades. Given a dataset $\mathcal{D}$, it is to mine the set of all rules from $\mathcal{D}$ such that each rule in Σ is above a threshold $\lambda = (\sigma, \delta)$ for its quality, measured in terms of support (*i.e.,* how frequent the rule can be applied to $\mathcal{D}$), and confidence (*i.e.,* how reliable the rule is for $\mathcal{D}$), which are controlled by configurable parameters σ and δ, respectively. We refer to such a discovery algorithm as a *batch algorithm* for mining the rules in a batch. A variety of batch discovery algorithms have been developed, *e.g.,* [2, 4, 6, 10–12, 20, 21, 23, 26, 28, 29, 35–39, 41, 42, 44, 48, 49, 54–56, 59–61, 65–67, 70, 71, 73–76, 78, 84, 87, 87–89, 91, 92, 94].

Rule discovery is, however, costly. For example, it takes 1.5 hours for the state-of-the-art (SOTA) BatchMiner [28] to mine REEs on a relation with 27 attributes and 5M tuples (Section 7). Moreover, setting good support and confidence thresholds $\lambda = (\sigma, \delta)$ is challenging, especially when users do

[*]Corresponding author.

Authors' Contact Information: Haoxian Chen, ShanghaiTech University, China, hxchen@shanghaitech.edu.cn; Wenfei Fan, Shenzhen Institute of Computing Sciences, China and University of Edinburgh, United Kingdom and Beihang University, China, wenfei@inf.ed.ac.uk; Jiaye Zheng, ShanghaiTech University, China, zhengjy2022@shanghaitech.edu.cn.

ACM 2836-6573/2025/6-ART175
https://doi.org/10.1145/3725312

not have sufficient prior knowledge about the datasets. For instance, when discovering rules for banking regulatory, if certain tables have relatively few records, setting a high support threshold could fail to discover meaningful rules. Conversely, a very low support threshold may lead to incorrect or unreliable rules, as they might be based on a limited number of instances. Additionally, if data instances are noisy, users often do not have precise knowledge about the extent of such noise, making it hard to determine an appropriate confidence threshold. Thus, in real-life applications, practitioners often have to adjust λ iteratively during the discovery process. As will be seen in Section 7, it takes 6 trials on average for a data quality expert to determine the right values for λ. However, it is too time-consuming to run a batch discovery algorithm for every λ value starting from scratch. One does not want to wait for hours when λ is slightly changed.

With this comes the need for studying incremental discovery in response to updates $\Delta\lambda$ to parameter λ. Suppose that a set Σ of rules has been found from dataset $\mathcal{D}$ subject to parameters λ by a batch algorithm $\mathcal{A}$. Informally, when λ is updated by $\Delta\lambda = (\Delta\sigma, \Delta\delta)$, incremental discovery aims to compute changes $\Delta\Sigma$ to Σ in response to $\Delta\lambda$ such that the set of rules discovered by $\mathcal{A}$ from $\mathcal{D}$ subject to $(\sigma + \Delta\sigma, \delta + \Delta\delta)$ is $\Sigma \oplus \Delta\Sigma$, where $\Sigma \oplus \Delta\Sigma$ means updating Σ with $\Delta\Sigma$. More specifically, a rule is in the updated set $\Sigma \oplus \Delta\Sigma$ if and only if it has support above $\sigma + \Delta\sigma$ and confidence at least $\delta + \Delta\delta$. The rational behind incremental discovery is that $\Delta\lambda$ is typically small in practice; when $\Delta\lambda$ is small, it is often more efficient to compute updates $\Delta\Sigma$ to Σ by minimizing unnecessary recomputation, than mining rules from $\mathcal{D}$ subject to $\lambda + \Delta\lambda$ starting from scratch.

Better yet, incremental algorithms in response to $\Delta\lambda$ suggest new paradigms for discovering rules. The users can start with parameter settings that are quick to discover an initial set of rules, and then gradually fine-tune σ and δ to mine rules that meet their need, based on the insights gained from the initial batch. Hence, practitioners may opt to discover the highest-quality rules first and then adjust σ and δ to find more. Alternatively, they may produce a stream of rule outputs with small delays between them, with no unnecessary recomputation and thus less total mining time (see Section 7).

No matter how desirable, little previous work has studied incremental rule discovery in response to parameter updates. Several issues need to be investigated. Is it possible to develop incremental algorithms that outperform batch discovery when $\Delta\lambda$ is frequent yet small? Moreover, can the algorithms guarantee the "effectiveness" of the incremental computation, *i.e.,* they warrant to incur the minimum cost when deducing the incremental algorithms from batch discovery? Can such algorithms scale with large datasets?

We answers these questions. As a testbed, we consider Rules for Entity Enhancing (REEs) [31, 33], which are used by Rock [1, 9] and deployed in a variety of real applications. Moreover, REEs subsume functional dependencies (FDs), conditional functional dependencies (CFDs) [25] and denial constraints (DCs) [8] as special cases. Thus an effective strategy for incremental REE discovery can be readily adapted to algorithms for mining all FDs, CFDs and DCs subject to support and confidence, not limited to BatchMiner of [28].

Challenges. The incremental discovery problem poses several challenges: (1) mining REEs across multiple tables with variables, beyond constant rules that are defined on a single tuple [7]; (2) determining the extent to which the incremental algorithm can effectively reuse auxiliary structures and intermediate results from batch discovery, with accuracy guarantees; (3) ensuring performance guarantees, including correctness, relative boundedness and parallel scalability (see below); (4) accelerating incremental discovery in practice, a challenging yet necessary task in industry, to accommodate updated parameters for complex rules such as REEs; and (5) mitigating the exponential growth of intermediate data sizes during the rule discovery process without compromising precision.

Contributions & organization. We study the incremental problem. After reviewing REEs in Section 2, we report the following.

(1) Problem statement (Section 3). We formulate three variants in response to $\Delta\sigma$, $\Delta\delta$ and $\Delta\lambda$ (*i.e.,* both $\Delta\sigma$ and $\Delta\delta$). We approach the problems by adapting incrementalization [34] of batch graph algorithms to relational rule discovery. Intuitively, when practitioners well understand how a batch discovery algorithm BatchMiner behaves to different inputs, we deduce incremental algorithms from it, retain the same logic and data structure of BatchMiner, and guarantee both the correctness and relative boundedness (see below).

(2) Incremental algorithm in response to $\Delta\sigma$ (Section 4). We develop an incremental discovery algorithm $\mathsf{IncMiner}_\sigma$ in response to updates $\Delta\sigma$ to the support threshold. We incrementalize the batch algorithm BatchMiner of [28], the SOTA algorithm for REE mining. We show that $\mathsf{IncMiner}_\sigma$ is (a) correct, *i.e.,* it computes exactly those rules to be removed or added when σ is increased ($\Delta\sigma > 0$) or decreased ($\Delta\sigma < 0$), respectively; and (b) bounded relative to its batch counterpart, *i.e.,* it incurs the minimum cost among all incrementalized algorithms of BatchMiner [32].

(3) Incremental algorithms in response to $\Delta\delta$ (Section 5). We also provide incremental discovery algorithms in response to updates $\Delta\delta$ to confidence threshold. The handling of $\Delta\delta$ is harder than its $\Delta\sigma$ counterpart since confidence does not have the anti-monotonicity. To speed up traversal of (possibly exponential) search lattice, we propose a sampling strategy and show that it is NP-complete to find an optimal sampling. This said, we develop (a) an exact algorithm $\mathsf{IncMiner}_\delta$ and (b) an approximate $\mathsf{IncMiner}_\delta^{\approx}$ with provable guarantees on the recall, both leveraging sampling. We show that both algorithms are bounded relative to BatchMiner [28].

(4) Incremental algorithms in response to both (Section 6). Putting $\mathsf{IncMiner}_\sigma$ and $\mathsf{IncMiner}_\delta$ together, we develop an incremental rule discovery algorithm $\mathsf{IncMiner}_\lambda$ in response to both $\Delta\delta$ and $\Delta\sigma$ at the same time. Moreover, we parallelize $\mathsf{IncMiner}_\lambda$, denoted by $\mathsf{PIncMiner}_\lambda$, to scale with large datasets. We show that relative to the batch algorithm BatchMiner [28, 63], $\mathsf{PIncMiner}_\lambda$ is not only bounded, but also parallelly scalable, *i.e.,* it guarantees to reduce parallel runtime when provided with more processors [51].

(5) Experimental study (Section 7). Using real-life data, we empirically find the following. (a) Incremental rule discovery is effective. $\mathsf{PIncMiner}_\lambda$ and $\mathsf{IncMiner}_{\delta(0.7)}^{\approx}$ consistently beat BatchMiner no matter whether $\Delta\sigma$ and $\Delta\delta$ are positive or negative, by up to 658×. (b) The incremental algorithms outperform BatchMiner even when $\Delta\sigma$ and $\Delta\delta$ account for 99% and 20% of σ and δ, respectively. (c) $\mathsf{PIncMiner}_\lambda$ is parallelly scalable. It is 4.3× faster when the number n of machines varies from 4 to 20. It is promising in practice. It takes 170*s* on a dataset with 32K tuples when $n = 20$, when $\Delta\sigma$ (resp. $\Delta\delta$) is 99% (resp. 10%) of σ (resp. δ), as opposed to 664*s* of the parallelized BatchMiner. (d) Our sampling and approximation strategies are effective, improving the performance by 4× and 9×, respectively.

Related work. We categorize the related work as follows.

Rule discovery. Discovery methods can be classified as follows (see [67, 84] for surveys): (1) levelwise lattice traversal for mining FDs [25, 39, 41, 44, 44, 56, 60, 61, 74, 76, 91, 92], association rules [35] and REEs [24, 27–29]; (2) depth-first search methods for FDs [2, 89] and DCs [10, 21, 55, 63]; (3) hybrid approaches for mining matching dependencies (MDs) [49, 73]; (4) learning-based methods for database dependencies [36, 94] and entity resolution (ER) rules [48, 75]; and (5) other technique such as tree-based search for frequent patterns [42] and association rules [88]. Parallel discovery methods have been developed for, *e.g.,* REEs [28, 29], under the Bulk Synchronous Parallel (BSP) [85] model (see [38] for a survey).

Rule mining has been widely used in various applications such as XAI [70], string matching [12], inductive logic programming [37, 59, 86], classification and regression [6, 11, 20, 54, 65, 66, 87].

As opposed to the prior work, we (a) study the problem of incremental rule discovery upon parameter changes; and (b) develop new lattice traversal and sampling methods to ensure the relative boundedness and minimize unnecessary recomputation. To do these, we incrementalize the (parallel) batch algorithm BatchMiner of [28].

Sampling. Sampling techniques have been widely employed in rule discovery to scale to large datasets, for association rules [16–19, 22, 43, 45, 46, 52, 53, 57, 58, 62, 69, 82, 93, 95], DCs [10, 55], FDs [50], REEs [28], and database queries [90]. Unlike the prior work where sampling is applied to the datasets, we sample the search lattice to enable efficient rule recovery upon parameter changes (Section 5).

Incremental rule discovery. The prior work has primarily focused on rule discovery in response to data updates: graph association rules in the presence of updates to graphs [30], DCs with tuple insertions [64], and point-wise order dependencies (PODs) with relation updates [79]. Incremental mining has also been studied for FDs [13–15, 74], association rules [5, 72, 81, 83], and temporal association rules [40] in response to database updates.

Closer to this work is IApriori [7], to mine constant association rules under dynamic thresholds (support and confidence) with the Apriori [4] algorithm. It extracts frequent itemsets from transaction dataset and reuses them in subsequent mining operations when thresholds are updated, thus reducing mining overhead.

As opposed to [5, 13–15, 30, 40, 64, 72, 74, 79, 81, 83], we study incremental discovery in response to updated parameters rather than updated datasets. Compared to IApriori [7], (a) we study REEs across different tables with multiple variables, beyond constant rules defined on a single tuple; (b) we not only reuse intermediate data structures but also incrementalize the mining algorithm; (c) we propose lattice search and sampling methods to deal with updated support and confidence, with accuracy guarantees; and (d) to our knowledge, this work provides the first incremental mining algorithms with the relative boundedness and parallel scalability.

2 Batch Discovery of REEs

This section first reviews Rules for Entity Enhancing (REEs) introduced in [9, 31] (Section 2.1). It then presents a SOTA batch algorithm for discovering REEs [28] (Section 2.2).

2.1 Collective Rules across Relations

We start with basic notations.

Preliminaries. We define REEs on a database schema $\mathcal{R} = (R_1, \ldots, R_m)$, where R_j is a relation schema $R_j(A_1 : \tau_1, \ldots, A_k : \tau_k)$, and each A_i is an attribute of type τ_i. An instance $\mathcal{D}$ of $\mathcal{R}$ is $(D_1, \ldots, D_m)$, where D_i is a relation of R_i, *i.e.,* a set of tuples of R_i ($i \in [1, m]$).

Predicates. *Predicates* over schema $\mathcal{R}$ are defined as follows:

$$p ::= R(t) \mid t.A \otimes c \mid t.A \otimes s.B,$$

where $\otimes$ is one of $=, \neq, <, \leq, >, \geq$. As in tuple relational calculus [3], (a) $R \in \mathcal{R}$, $R(t)$ is a *relation atom* of $\mathcal{R}$, and t is a *tuple variable bounded by* $R(t)$; (b) $t.A$ is an attribute of t if t is bounded by $R(t)$ and A is an attribute in R; (c) $t.A \otimes c$ is *a constant predicate* if c is a value in the domain of A; and (d) in $t.A \otimes s.B$, tuple t (resp. s) is bounded by $R(t)$ (resp. $R'(s)$), and $A \in R$ and $B \in R'$ have the same type.

REEs. A *rule for entity enhancing* (REE) φ over schema $\mathcal{R}$ is

$$\varphi : X \rightarrow p_0,$$

where X is a conjunction of *predicates* over $\mathcal{R}$, and p_0 is a predicate over $\mathcal{R}$ whose tuple variables also appear in X. We refer to X as the *precondition* of φ, and p_0 as the *consequence* of φ.

Example 1: Consider a person relation with attributes id, country, area-code, city, Mstatus (marital status), citizen, and the year when the tuple is recorded. Below are example REEs on person.

(1) φ_1 = person(t_1) $\wedge$ person(t_2) $\wedge$ t_1.citizen = "US" $\wedge$ t_2.citizen = "Norway" $\rightarrow$ t_1.id $\neq$ t_2.id. It says that no one can be a citizen of both the US and Norway, because Norway does not admit dual citizenship. The rule helps us decide whether two persons match or not.

(2) φ_2 = person(t_1) $\wedge$ person(t_2) $\wedge$ t_1.id = t_2.id $\wedge$ t_1.Mstatus = "single" $\wedge$ t_2.Mstatus = "Married" $\rightarrow$ t_1.year $\leq$ t_2.year. Intuitively, this rule says that the marital status of a person can change from single to married, but not the other way around.

(3) φ_3 = person(t) $\wedge$ t.country = "US" $\wedge$ t.area$-$code = 215 $\rightarrow$ t.city = "Philly". It states the binding between area-code and city.

(4) φ_4 = person(p_1) $\wedge$ person(p_2) $\wedge$ award(a_1) $\wedge$ award(a_2) $\wedge$ p_1.id = a_1.id $\wedge$ p_2.id = a_2.id $\wedge$ a_1.year = a_2.year $\wedge$ a_1.award = "Golden Bear" $\wedge$ a_2.award = "Gold Lion" $\rightarrow$ p_1.id $\neq$ p_2.id. It says that no one won both film awards in the same year. It involves four tuple variables across two tables (person and award). □

Semantics. Consider an instance $\mathcal{D}$ of $\mathcal{R}$. A *valuation h* of tuple variables of $\varphi = X \rightarrow p_0$ in $\mathcal{D}$, or simply a valuation of φ, is a mapping that instantiates each variable t of φ with a tuple in $\mathcal{D}$.

We say that h *satisfies* a predicate p, written as $h \models p$, by following the standard semantics of first-order logic as in tuple relational calculus [3]; *e.g.,* if $p = t.A < s.B$, h maps t to tuples t_1 and s to t_2, and $t_1.A < t_2.B$, then $h \models p$. For precondition X, $h \models X$ if $h \models p$ for *all* predicates p in X. We write $h \models \varphi$ if $h \models X$ implies $h \models p_0$.

An instance $\mathcal{D}$ of $\mathcal{R}$ *satisfies* φ, denoted by $\mathcal{D} \models \varphi$, if *for all* valuations h of tuple variables of φ in $\mathcal{D}$, $h \models \varphi$. We say that $\mathcal{D}$ *satisfies* a set Σ of REEs, denoted by $\mathcal{D} \models \Sigma$, if for all $\varphi \in \Sigma$, $\mathcal{D} \models \varphi$.

Remark. (1) In the general definition of REEs [31], machine learning (ML) models $\mathcal{M}$ can be embedded as predicates, as long as the models return a Boolean. To simplify the discussion, here we consider REEs without ML predicates. (2) The REEs considered in this paper subsume CFDs and DCs as special cases [31, 33].

2.2 A Batch Algorithm for Rule Discovery

We next review batch discovery of REEs and its algorithm.

Support and confidence. We want to discover high-quality REEs. The quality of rules is typically measured by the two criteria below.

Support. This is to quantify how often an REE $\varphi = X \rightarrow p_0$ can be applied to a dataset $\mathcal{D}$. We define the *support* of φ in $\mathcal{D}$ as [29]:

$$\text{supp}(\varphi, \mathcal{D}) = |\text{spset}(\varphi, \mathcal{D})|,$$

where spset$(\varphi, \mathcal{D})$ is the set of all tuple pairs $\langle h(t_0), h(s_0)\rangle$ such that h is a valuation of φ in $\mathcal{D}$, t_0 and s_0 are tuple variables in p_0, $h \models X$ and $h \models p_0$. Note that spset$(\varphi, \mathcal{D})$ is defined in terms of tuples that instantiate the consequence p_0 of φ, which is either a binary or unary predicate. This definition generalizes support for CFDs, which are restricted to rules in a single table, whereas REEs span multiple tables. While REE support can adapt to CFDs, the reverse does not hold. To simplify the discussion, we consider binary p_0; the notion of spset$(\varphi, \mathcal{D})$ can be readily extended to unary p_0.

It is known that supp$(\varphi, \mathcal{D})$ has *the anti-monotonicity property* [29]. Given REEs $\varphi : X \rightarrow p_0$ and $\varphi' : X' \rightarrow p_0$ with the same consequence p_0, we write $\varphi \preceq \varphi'$ if $X \subseteq X'$, *i.e.,* φ is less restrictive than φ'. Then for any instance $\mathcal{D}$ of $\mathcal{R}$ and REEs φ and φ', if $\varphi \preceq \varphi'$, then spset$(\varphi', \mathcal{D})$

$\subseteq$ spset$(\varphi, \mathcal{D})$ and supp$(\varphi', \mathcal{D}) \leq$ supp$(\varphi, \mathcal{D})$.

Confidence. It measures how strong the association between precondition X and consequence p_0 is for an REE $\varphi = X \longrightarrow p_0$. More specifically, the *confidence* of φ on $\mathcal{D}$ is a value in [0, 1] defined as:

$$\mathsf{conf}(\varphi, \mathcal{D}) = \frac{|\mathsf{spset}(\varphi, \mathcal{D})|}{|\mathsf{spset}(X, \mathcal{D})|},$$

where spset$(X, \mathcal{D})$ is the set of tuple pairs satisfying all predicates in X. The use of confidence helps us tolerate noise, such that useful rules could still be discovered from the noisy data.

For an integer σ, an REE is *σ-frequent* on $\mathcal{D}$ if supp$(\varphi, \mathcal{D}) \geq \sigma$. For a threshold δ, REE φ is *δ-confident* on $\mathcal{D}$ if conf$(\varphi, \mathcal{D}) \geq \delta$. In the sequel, we write conf$(\varphi, \mathcal{D})$ and supp$(\varphi, \mathcal{D})$ as conf(φ) and supp(φ), respectively, when $\mathcal{D}$ is clear from the context.

Minimality. An REE $\varphi : X \longrightarrow p_0$ over R is considered trivial if $p_0 \in X$. We focus solely on non-trivial REEs.

An REE $\varphi : X \longrightarrow p_0$ is *left-reduced* on D if φ is σ-frequent and δ-confident, and there exists no REE φ' such that $\varphi' \preceq \varphi$ and φ' is also σ-frequent and δ-confident. In other words, no predicate in X can be removed, *i.e.,* the predicates are minimal.

A *minimal* REE φ on D is a non-trivial and left-reduced REE.

Cover. Denote by $\Sigma_{(\sigma,\delta)}$ the set of all σ-frequent and δ-confident REEs on dataset $\mathcal{D}$. The set often includes trivial rules, redundant predicates in X and even redundant rules that are entailed by the other rules in $\Sigma_{(\sigma,\delta)}$. Such rules make $\Sigma_{(\sigma,\delta)}$ excessively large. To remove such useless rules, we use the following notions.

A set Γ of REEs *implies* an REE φ, denoted by $\Gamma \models \varphi$, if for any dataset $\mathcal{D}$ of database schema $\mathcal{R}$, whenever $\mathcal{D} \models \Gamma$, then $\mathcal{D} \models \varphi$, *i.e.,* φ is a logical consequence of Γ and is hence "redundant".

A *cover* Σ *of* $\Sigma_{(\sigma,\delta)}$ is a subset of $\Sigma_{(\sigma,\delta)}$ such that (a) for each $\varphi \in \Sigma_{(\sigma,\delta)}$, $\Sigma \models \varphi$, *i.e.,* Σ and $\Sigma_{(\sigma,\delta)}$ are logically "equivalent"; (b) for each $\varphi \in \Sigma$, $\Sigma \setminus \{\varphi\} \not\models \varphi$, *i.e.,* no REE in Σ can be entailed by the other rules in Σ; in other words, no REEs in Σ are redundant.

The batch discovery problem. The problem is stated as follows.

- *Input*: A schema $\mathcal{R}$, an instance $\mathcal{D}$ of $\mathcal{R}$, and parameter $\lambda = (\sigma, \delta)$.
- *Output*: A cover Σ of the set $\Sigma_{(\sigma,\delta)}$ of all REEs that are both σ-frequent and δ-confident on dataset $\mathcal{D}$.

Here σ is a positive integer and $\delta \in [0, 1]$. We refer to Σ as *the set of λ-bounded* REEs *on* $\mathcal{D}$. Intuitively, the batch discovery problem aims to mine all high-quality REEs on dataset $\mathcal{D}$ subject to predefined thresholds σ and δ for support and confidence, respectively.

Batch algorithm. We present the batch discovery algorithm in Figure 1, referred to as BatchMiner. As a SOTA in REE mining [28], BatchMiner generates candidate REEs φ levewisely, adding one predicate at a time. A costly step is to compute the support and confidence of φ and check whether they are above δ and σ, respectively.

BatchMiner takes as input samples $\mathcal{D}_s$ of dataset $\mathcal{D}$, two sets RHS and P_0 of predicates, and thresholds for support σ and confidence δ. It is to discover a set Σ of REEs such that for each $\varphi = X \longrightarrow p_0$ in Σ, (1) $p_0 \in$ RHS, (2) $X \subseteq \mathsf{P}_0$ and (3) φ is σ-frequent and δ-confident. Intuitively, for an application, RHS is the set of consequence predicates of users' interest, and P_0 is the set of predicates correlated to those in RHS, which can be identified via reinforcement learning (see [28] for details). That is, BatchMiner discovers only those REEs relevant to the application.

BatchMiner initializes an empty set Σ. It builds position list indexes (PLI) [63] to speed up support and confidence computation. For each distinct value Val of an attribute A in relation R, PLI maintains a list of positions in the database where that value occurs: $(R, A, \mathsf{Val}) \mapsto \mathsf{List}[\mathsf{Index}]$. Then for each

```
Algorithm BatchMiner
Input: $\mathcal{R}$, $\mathcal{D}_s$, RHS, $P_0$, $\sigma$ and $\delta$.
Output: A cover $\Sigma_c$ of the set of minimal $\sigma$-frequent and $\delta$-confident REEs such that for each $\varphi : X \rightarrow p_0$ in
    $\Sigma_c$, (1) $p_0 \in$ RHS; and (2) $X \subseteq \mathsf{P}_0$.
1.  $\Sigma := \emptyset$;
2.  Build position list indexes (PLI) ;
3.  for each $p_0 \in$ RHS do
4.      $\mathsf{P}_{\mathsf{sel}} := \emptyset$;  $\mathsf{P}_{\mathsf{re}} := \mathsf{P}_0$;
5.      $\Sigma :=$ Expand($\mathcal{D}_s, \mathsf{P}_{\mathsf{sel}}, \mathsf{P}_{\mathsf{re}}, p_0, \delta, \sigma, \Sigma$);
6.  $\Sigma_c :=$ computeCover($\Sigma$);
7.  return $\Sigma_c$;

Procedure Expand
Input: $\mathcal{D}_s, \mathsf{P}_{\mathsf{sel}}, \mathsf{P}_{\mathsf{re}}, p_0, \delta, \sigma$ and the current set $\Sigma$ of minimal REEs.
Output: An updated set $\Sigma$ of minimal REEs.
8.  $Q :=$ an empty queue;  $Q$.add($\langle \mathsf{P}_{\mathsf{sel}}, \mathsf{P}_{\mathsf{re}} \rangle$);
9.  while $Q \neq \emptyset$ do
10.     $\langle \mathsf{P}_{\mathsf{sel}}, \mathsf{P}_{\mathsf{re}} \rangle := Q$.pop();  $\varphi := \mathsf{P}_{\mathsf{sel}} \rightarrow p_0$;
11.     if $\varphi$ is minimal then
12.         $\Sigma := \Sigma \cup \{\varphi\}$;
13.         continue; // do not further expand
14.     if supp($\varphi$) $\geq \sigma$ then  // Anti-monotonicity
15.         for each $p \in \mathcal{P}_{\mathsf{re}}$ do // Add predicates from $\mathsf{P}_{\mathsf{re}}$ to $\mathsf{P}_{\mathsf{sel}}$
16.             $Q$.add($\langle \mathsf{P}_{\mathsf{sel}} \cup \{p\}, \mathsf{P}_{\mathsf{re}} \setminus \{p\} \rangle$);
17. return $\Sigma$;
```

Fig. 1. Algorithm BatchMiner

p_0 in RHS, procedure Expand is invoked to discover REEs that have p_0 as the consequence, and add the mined REEs into Σ. Finally, the cover Σ_c of Σ is computed by eliminating redundant REEs in Σ.

Procedure Expand updates the set Σ with *minimal* REEs φ for consequence p_0. It begins by initializing an empty queue Q and adding the pair $\langle \mathsf{P}_{\mathsf{sel}}, \mathsf{P}_{\mathsf{re}} \rangle$ to Q. While Q is not empty, it pops a pair $\langle \mathsf{P}_{\mathsf{sel}}, \mathsf{P}_{\mathsf{re}} \rangle$ and forms an REE $\varphi = \mathsf{P}_{\mathsf{sel}} \rightarrow p_0$. If φ is minimal, it is added to Σ, and Expand continues to the next iteration. If φ has support above σ, new candidate REEs are created by moving one predicate p from P_{re} to $\mathsf{P}_{\mathsf{sel}}$, and Expand recursively mines longer REEs with larger confidence. If none of the two conditions is met, the current search branch can be safely discarded by the anti-monotonicity of REE support. The iteration continues until the queue Q becomes empty, ensuring that all relevant REEs are explored and added to Σ. Finally, the updated set Σ is returned.

Complexity. BatchMiner runs in exponential time in the worst case. Indeed, there may be exponentially many REEs in the size of $\mathcal{D}$. However, optimization techniques, such as the space pruning employed in BatchMiner and other strategies [24, 28, 29], have made it practical for real-world application [9], for schema design and data cleaning. It is to further speed up the batch mining process that we will develop incremental algorithms in Sections 4–6.

3 Incremental Discovery Problems

This section formulates the incremental rule discovery problems in response to parameter updates, and presents an incrementalization approach towards the problems. We start with basic notations.

The incremental discovery problems. As remarked earlier, practitioners often need to adjust parameters σ and δ in order to discover rules that meet their practical needs. In light of this, we want to incrementally compute changes to Σ in response to adjusted support and confidence. There are in fact three problems here.

- *Input*: Database schema $\mathcal{R}$, dataset $\mathcal{D}$ of $\mathcal{R}$, $\lambda = (\sigma, \delta)$, the set Σ of λ-bounded REEs on $\mathcal{D}$, and a (positive or negative) integer $\Delta\sigma$.
- *Output*: Updates $\Delta_\sigma\Sigma = (\Delta\Sigma_+, \Delta\Sigma_-)$ to Σ such that the set of λ_σ-bounded REEs on $\mathcal{D}$ is $\Sigma \oplus \Delta_\sigma\Sigma$, where $\lambda_\sigma = (\sigma+\Delta\sigma, \delta)$, and $\Delta\Sigma_+$ (resp. $\Delta\Sigma_-$) includes REEs to be added to (resp. removed from) Σ.

When the support threshold is changed to $\sigma + \Delta\sigma$, the problem is to compute updates $\Delta_\sigma\Sigma$ to Σ such that $\Sigma \oplus \Delta_\sigma\Sigma$ is the set of $(\sigma + \Delta\sigma, \delta)$-bounded REEs on $\mathcal{D}$, by reusing Σ as much as possible, without recomputing all the $(\sigma + \Delta\sigma, \delta)$-bounded REEs from scratch. Here $\Delta\sigma$ can be either positive or negative; $\Delta\Sigma_+$ includes newly added REEs with smaller supports if $\Delta\sigma < 0$, and $\Delta\Sigma_-$ consists of "low-support" REEs removed from Σ if $\Delta\sigma > 0$.

Example 2: Continuing with Example 1, where the REEs are mined following an initial support $\sigma = 100$ and confidence $\delta = 80\%$. After analyzing REEs, the user decides to lower the support threshold σ to 50 to discover more rules. One of the newly mined REE is $\varphi_5 = \text{person}(t_1) \wedge \text{person}(t_2) \wedge t_1.\text{citizen} = \text{"US"} \wedge t_2.\text{citizen} = \text{"Japan"} \rightarrow t_1.\text{id} \neq t_2.\text{id}$. Similar to φ_1, this rule says that a person cannot be a citizen of both the US and Japan, as Japan does not admit dual citizenship either. However, as there are fewer records from Japan in the dataset, φ_5 was not mined until σ is lowered. □

Similarly, we study the incremental problem in response to $\Delta\delta$.

- *Input*: $\mathcal{R}$, $\mathcal{D}$, $\lambda = (\sigma, \delta)$ and Σ as above, and a number $\Delta\delta \in [-1, 1]$ such that $\delta + \Delta\delta \in [0, 1]$.
- *Output*: The updates $\Delta_\delta\Sigma = (\Delta\Sigma_+, \Delta\Sigma_-)$ to Σ such that the set of λ_δ-bounded REEs on $\mathcal{D}$ is $\Sigma \oplus \Delta_\delta\Sigma$, where $\lambda_\delta = (\sigma, \delta + \Delta\delta)$.

Here $\Delta\Sigma_-$ includes low-confidence REEs to be removed from Σ ($\Delta\delta > 0$), and $\Delta\Sigma_+$ collects REEs to be added ($\Delta\delta < 0$).

We also study the problem in response to both $\Delta\sigma$ and $\Delta\delta$.

- *Input*: $\mathcal{R}$, $\mathcal{D}$, $\lambda = (\sigma, \delta)$ and Σ as above, a (positive or negative) integer $\Delta\sigma$, and a number $\Delta\delta \in [-1, 1]$ such that $\delta + \Delta\delta \in [0, 1]$.
- *Output*: The updates $\Delta_\lambda\Sigma = (\Delta\Sigma_+, \Delta\Sigma_-)$ to Σ such that the set of λ'-bounded REEs on $\mathcal{D}$ is $\Sigma \oplus \Delta_\delta\Sigma$, where $\lambda' = (\sigma + \Delta\sigma, \delta + \Delta\delta)$.

Incrementalization. We approach the problems above by following the incrementalization approach of [32, 34]. Incrementalization is to pick a batch discovery algorithm $\mathcal{A}$ that has been verified effective, and deduce an incremental algorithm $\mathcal{A}_\Delta$ from $\mathcal{A}$, by *reusing the original logic and data structures of* $\mathcal{A}$ as much as possible.

More formally, denote an instance of the rule discovery problem as $I = (\mathcal{D}, \lambda)$, and the set of λ-bounded REEs discovered by batch algorithm $\mathcal{A}$ on dataset $\mathcal{D}$ as $\mathcal{A}(I)$. The incremental algorithm $\mathcal{A}_\Delta$ is deduced from $\mathcal{A}$ with the following guarantees:

(1) Correctness: Given an instance I and updates ΔI to I, it computes updates $\Delta\Sigma$ to the output $\mathcal{A}(I)$ such that $\mathcal{A}(I \oplus \Delta I) = \mathcal{A}(I) \oplus \Delta\Sigma$, which is precisely the new output of $\mathcal{A}$ on the updated input $I \oplus \Delta I$. Here ΔI denotes updates to the parameter (*i.e.,* $\Delta\lambda$).

(2) Efficiency: Algorithm $\mathcal{A}_\Delta$ is *bounded relative to* $\mathcal{A}$ [32]. That is, the size of the data inspected by $\mathcal{A}_\Delta$ is a function in the size $|\mathsf{AFF}|$ of *the affected area* AFF, not in (possibly big) $|\mathcal{D}|$. To see how AFF is defined, for an instance $I = (\mathcal{D}, \lambda)$ of the discovery problem, denote by $I_{(\mathcal{A},\lambda)}$ the data accessed by $\mathcal{A}$ for discovering the set Σ of λ-bounded REEs, including the part of $\mathcal{D}$ inspected by $\mathcal{A}$, the collection of candidate REEs generated, and auxiliary structure used by $\mathcal{A}$. Then for updates ΔI to I, AFF denotes the difference between $(I \oplus \Delta I)_{(\mathcal{A},\lambda)}$ and $I_{(\mathcal{A},\lambda)}$, *i.e.,* the difference in the data inspected by the batch algorithm $\mathcal{A}$ for computing $\mathcal{A}(I \oplus \Delta I)$ and $\mathcal{A}(I)$.

Intuitively, AFF is the part of the data that is necessarily checked by the batch algorithm $\mathcal{A}$ in

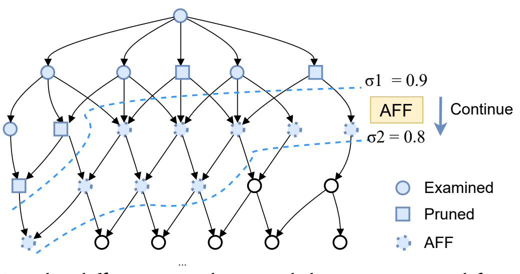


Fig. 2. AFF on σ changes, *i.e.,* the difference in the search lattice examined for support thresholds σ_1 and σ_2.

response to ΔI, and hence, |AFF| is the inherent updating cost of incrementalizing $\mathcal{A}$. When $|\Delta I|$ is small, |AFF| is often small as well, and thus $\mathcal{A}_\Delta$ is often faster than $\mathcal{A}$; in other words, $\mathcal{A}_\Delta$ aims to minimize unnecessary recomputation. Moreover, when practitioners get used to algorithm $\mathcal{A}$ and understand how it behaves *w.r.t.* different inputs, they want to stick to it. Thus we incrementalize $\mathcal{A}$ and retain its logic and data structures.

4 Incremental Algorithm for $\Delta\sigma$

This section develops an incremental discovery algorithm for REEs in response to support updates $\Delta\sigma$, denoted by IncMiner_σ. We develop IncMiner_σ by incrementalizing BatchMiner following [32, 34], such that IncMiner_σ is bounded relative to BatchMiner.

We start by identifying the affected area AFF.

AFF. Consider the difference between the set of candidate REEs explored by BatchMiner given two different support thresholds, σ and $\sigma + \Delta\sigma$, denoted by $\Sigma_{\Delta\sigma}$. When $\Delta\sigma < 0$, $\Sigma_{\Delta\sigma}$ is defined as:

$$\Sigma_{\Delta\sigma} = \{\varphi \mid \sigma + \Delta\sigma \leq \text{supp}(\varphi) \leq \sigma\}. \quad (1)$$

This is evident in BatchMiner, where σ is examined at line 14 (Figure 1). Here, candidate REEs φ with support below σ are discarded by the anti-monotonicity of support. By lowering the support threshold to $\sigma + \Delta\sigma$, we continue exploring these REEs by adding more predicates to φ (line 15). The additional REEs explored by it constitute precisely $\Sigma_{\Delta\sigma}$. For $\Delta\sigma > 0$, $\Sigma_{\Delta\sigma}$ is defined similarly.

Figure 2 visualizes this. It constitutes a search lattice, where each node is an examined REE, and an edge $\varphi_a \longrightarrow \varphi_b$ means that φ_b is a direct expansion of φ_a (by adding one predicate to its precondition X). Given two support thresholds $\sigma_1 > \sigma_2$, denote by Σ_1 and Σ_2 the set of REEs examined by BatchMiner *w.r.t.* σ_1 and σ_2, respectively. By the anti-monotonicity of REE support, $\Sigma_1 \subseteq \Sigma_2$. AFF thus includes the difference $\Sigma_{\Delta\sigma} = \Sigma_2 \setminus \Sigma_1$, as highlighted by the set of nodes between the two horizontal dashed lines in Figure 2.

In addition, AFF includes the portion of data $\mathcal{D}$ examined during the computation of support and confidence for each φ in $\Sigma_{\Delta\sigma}$. Since $\mathcal{D}$ is preprocessed into a PLI, we define AFF in terms of PLI:

$$\text{AFF}_{\Delta\sigma} = \{P(\varphi) \mid \varphi \in \Sigma_{\Delta\sigma}\} \quad (2)$$

Here $P(\varphi)$ is the subset of PLI inspected by the batch algorithm BatchMiner for computing φ's support and confidence.

Deducing algorithm IncMiner_σ. We next develop IncMiner_σ, shown in Figure 3, using feasible states and auxiliary structures.

Feasible states. As shown in Figure 1, BatchMiner primarily operates by iteratively updating two state variables: the set Σ of discovered λ-bounded REEs and the set Q of candidate REEs to be further expanded. Denote by f the update procedure outlined from lines 10 to 16 in Figure 1. BatchMiner continuously applies f to Σ and Q until Q becomes empty, at which point it returns Σ_c as the output.

Upon support update $\Delta\sigma$, an incrementalization strategy is to reset the state variables Σ and Q

```
Input: Database schema $\mathcal{R}$, dataset $\mathcal{D}$, $\lambda = (\sigma, \delta)$, the set $\Sigma$ of $\lambda$-bounded REEs on $\mathcal{D}$, the set of pruned REEs
   in prior mining process $\Sigma_{<\sigma}$, a (positive or negative) integer $\Delta\sigma$.
Output: The set $\Delta\Sigma_+$ (resp. $\Delta\Sigma_-$) of REEs to be added to (resp. removed from) $\Sigma$, and the set of pruned REEs
   $\Sigma_{<\sigma'}$ with support lower than $\sigma + \Delta\sigma$.
1. if $\Delta\sigma > 0$ then
2.    return $(\emptyset, \{\varphi \in \Sigma \mid \mathsf{supp}(\varphi) < \sigma + \Delta\sigma\}, \Sigma_{<\sigma})$;
3. else  // $\Delta\sigma < 0$
4.    $\Delta\Sigma_+ := \emptyset$;  $\Sigma_{<\sigma'} := \Sigma_{<\sigma}$;
5.    for each $p_0 \in$ RHS do
6.       $Q := \{\varphi \in \Sigma_{<\sigma} \mid \mathsf{supp}(\varphi) \geq \sigma + \Delta\sigma \wedge \varphi.p_0 = p_0\}$;
7.       $(\Delta\Sigma_+, \Sigma_{<\sigma'}) := \mathsf{IncExpand}(\mathcal{D}, Q, p_0, \delta, \sigma + \Delta\sigma, \Delta\Sigma_+, \Sigma_{<\sigma'})$;
8. return $(\Delta\Sigma_+, \emptyset, \Sigma_{<\sigma'})$;

Procedure IncExpand
Input: $\mathcal{D}_s, Q_0, p_0, \delta, \sigma$, the current set $\Sigma$ of $\lambda$-bounded REEs, and the current set $\Sigma_{<\sigma}$ of pruned REEs.
Output: An updated set $\Sigma$ of minimal REEs and updated $\Sigma'_{<\sigma}$.
9. $Q := Q_0$
10. while $Q \neq \emptyset$ do
       ... // same as lines 10-13 in Figure 1.
11.       if $\mathsf{supp}(\varphi) \geq \sigma$ then
             ... // update Q as in lines 15-16 in Figure 1.
12.       else // supp $(\varphi) < \sigma$
13.          $\Sigma'_{<\sigma} := \Sigma_{<\sigma} \cup \{\varphi\}$;
14. return $(\Sigma, \Sigma'_{<\sigma})$;
```

Fig. 3. Algorithm IncMiner$_\sigma$

into feasible states aligned with the updated support parameter. Then the solution can be obtained by iteratively applying an update procedure f_Δ, which reuses most part of the step function f in BatchMiner, to Σ and Q.

Denote by Σ^t and Q^t the status of variable Σ and Q at the t^{th} iteration, respectively. An element φ in Σ^t is *feasible* if it is λ-bounded.

An element $(P_{\mathsf{sel}}, P_{\mathsf{re}})$ in Q^t is *feasible* if (1) $\mathsf{supp}(P_{\mathsf{sel}} \rightarrow p_0) \geq \sigma$; and (2) there exists no subset $X' \subset P_{\mathsf{sel}}$ such that $\mathsf{conf}(X' \rightarrow p_0) \geq \delta$. Here condition (2) ensures the minimality of the mined REEs. We say that state variable Σ^t (resp. Q^t) is *feasible* if all elements in Σ^t (resp. Q^t) are in a feasible state.

Auxiliary data structures. In order to recover the search queue Q into a feasible state, IncMiner$_\sigma$ keeps the set of candidate REEs that are pruned due to insufficient support, denoted as $\Sigma_{<\sigma}$. Indeed, these pruned REEs would become *valid*, *i.e.,* they may meet a smaller support threshold (when $\Delta\sigma < 0$) and should be explored.

IncMiner$_\sigma$. Using feasible elements in Σ and Q, and auxiliary structures $\Sigma_{<\sigma}$, IncMiner$_\sigma$ separates two cases as shown in Figure 3.

(1) When $\Delta\sigma > 0$. By the λ-boundedness and minimality guarantee of BatchMiner, each REE φ in the output Σ of BatchMiner with $\mathsf{supp}(\varphi) \geq \sigma + \Delta\sigma$ is also λ_σ-bounded and minimal. Thus, the set $\Delta\Sigma$ of minimal λ_σ-bounded REEs can be obtained by directly filtering REEs in Σ (line 2), without the need for further expanding.

(2) When $\Delta\sigma < 0$, REEs pruned in prior mining with lower σ might become a feasible element in the search queue. Thus, these rules are put into Q (line 6). IncMiner$_\sigma$ incrementalizes procedure *Expand* to IncExpand, reusing most steps in Expand (lines 10-16 in Figure 1), except that at lines

12-13, upon discovering REEs with support lower than the threshold σ, instead of discarding them as in BatchMiner, these REEs are put into the auxiliary $\Sigma'_{<\sigma}$ for incremental discovery in future updates.

Example 3: Continuing with Example 2, Figure 2 illustrates the flow of IncMiner_σ. When support $\sigma = 100$, some incomplete REEs might get pruned due to insufficient support, *e.g.,* $\varphi'_5 = \text{person}(t_1) \wedge \text{person}(t_2) \wedge t_2.\text{citizen} = \text{“Japan”} \rightarrow t_1.\text{id} \neq t_2.\text{id}$; it was pruned due to its low support at 80. By IncMiner_σ, φ'_5 is put into $\Sigma_{<\sigma}$.

When σ is reduced to 50, IncMiner_σ starts with the set of pruned REEs whose support lies between 50 and 100 (marked as square boxes), and continues expanding them until all newly qualified ones are found or the support drops below the new $\sigma = 50$. Note that the newly examined REEs are exactly those in AFF defined above.

By IncExpand in Figure 3, new REEs are mined by adding new predicates to REEs in $\Sigma_{<\sigma}$. For instance, adding $t_1.\text{citizen} = \text{“US”}$ to the precondition of φ'_5 yields φ_5 in Example 2. Hence, incremental mining can uncover additional rules by lowering threshold σ. □

Correctness. When $\Delta\sigma > 0$, *i.e.,* support threshold increases, no further search is needed, because by the anti-monotonicity of REE support, the newly feasible REEs are already a subset of previous mining results, *i.e.,* $\Sigma_{(\sigma+\Delta\sigma,\delta)} \subseteq \Sigma_{(\sigma,\delta)}$. On the other hand, when $\Delta\sigma < 0$, *i.e.,* support threshold decreases, IncMiner_σ is a natural continuation of its batch counterpart BatchMiner. It resumes the search by putting all the newly feasible candidate REEs from $\Sigma_{<\sigma}$ into the search queue Q. Starting from the reset queue Q, the set of REEs explored by IncMiner_σ is precisely $\Sigma_{\Delta\sigma}$ (Equation 1).

Relative boundedness. One can see that IncMiner_σ inspects only those candidate REEs in $\Sigma_{\Delta\sigma}$ and their associated data (including auxiliary structures), which are confined in the affected area AFF by $\Delta\sigma$ (see AFF above). Moreover, the computation of cover conducts necessary work for updated set of REEs, which involves only REE implication but not dataset $\mathcal{D}$; it has to be performed by any incrementalization of BatchMiner. Putting these together, one can verify that IncMiner_σ is bounded relative to BatchMiner, *i.e.,* its time cost is measured by a function in |AFF|, not in possibly big $|\mathcal{D}|$.

Space overhead. The additional intermediate states $\Sigma_{<\sigma}$ is exponential in the predicate space $|P|$, denoted as $O(2^{|P|})$ (the set of all REEs constructed using predicates in P). This is the same as the space complexity of BatchMiner, since its output, *i.e.,* the set Σ of λ-bounded REEs, is also in $O(2^{|P|})$. This said, IncMiner_σ is much faster than BatchMiner in practice as will be seen in Section 7.

5 Incremental Algorithm for $\Delta\delta$

This section develops an incremental REE discovery algorithm, denoted by IncMiner_δ, in response to updates to confidence threshold δ. The algorithm is more challenging than its counterpart for $\Delta\sigma$ since as opposed to support, confidence does not have the anti-monotonicity. To take up the challenge, we first develop a sampling strategy to reduce the search space (Section 5.1). We then deduce IncMiner_δ and show its relative boundedness (Section 5.2).

5.1 Sampling for Search Lattice

Note that in Algorithm 1, the search for a candidate REE φ terminates under conditions: (1) φ is minimal λ-bounded; or (2) φ has support lower than σ. Denote by $\Sigma_{(\sigma,\delta)}$ the set of (σ,δ)-bounded REEs. Now suppose that the confidence threshold δ is increased to δ' ($\Delta\delta > 0$). Then some of the REEs in $\Sigma_{(\sigma,\delta)}$ may no longer be qualified, and a continuation of the search down the lattice has to be performed until either termination condition is met. As σ is unchanged, lattice pruned by condition 2 (insufficient support) remains pruned. Thus only the successors of REEs in $\Sigma_{(\sigma,\delta)}$ are expanded.

AFF. To see AFF, let $\varphi_b \geq \varphi_a$ denote that φ_b is a successor of φ_a in the search lattice. Then we

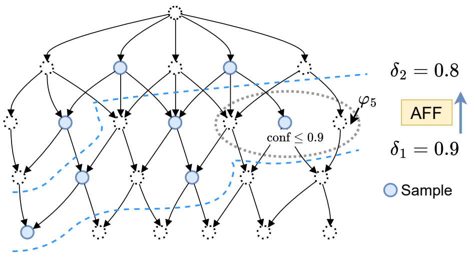


Fig. 4. Sample search lattice. The full lattice can be recovered by enumerating neighbors of each sampled REE.

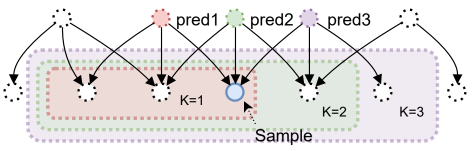


Fig. 5. Sample coverage with different radius K.

id	country	area-code	city	Mstatus	citizen	year
t_1	United States	840	New York	married	US	2010
t_2	United States	840	Boston	married	US	2020
t_3	United States	840	Boulder	single	US	2012
t_4	Japan	392	New York	single	US	2015
t_5	Japan	392	Tokyo	single	Japan	2018

Table 1. An example of Person relation

define the difference of REEs examined by two confidence thresholds δ and $\delta' = \delta + \Delta\delta$ as follows:

$$\begin{aligned}\Sigma_{\succeq\delta} &= \{\varphi \mid \varphi \geq \varphi' \wedge \varphi' \in \Sigma_{(\sigma,\delta)}\},\\ \Sigma_{\Delta\delta} &= \{\varphi \in \Sigma_{\succeq\delta} \mid \mathsf{conf}(\varphi) \leq \delta' \wedge \mathsf{supp}(\varphi) \geq \sigma\}.\end{aligned}$$

Here $\Sigma_{\succeq\delta}$ includes all successors of λ-bounded REEs in $\Sigma_{(\sigma,\delta)}$, and $\Sigma_{\Delta\delta}$ denotes the precise difference of REEs explored by BatchMiner with different confidence thresholds δ and δ': successors of previously mined REEs are only examined if they have support above σ and confidence below δ'. The set of difference in REEs is defined symmetrically for $\Delta\delta < 0$. Figure 4 visualizes the AFF for the case where δ is decreased from 0.9 to 0.8, where the two horizontal dashed lines represent the search frontier when $\delta = 0.8$ and 0.9, respectively. Thus $\Sigma_{\Delta\delta}$ is the area between the two lines.

Similar to AFF for support updates, AFF for confidence updates can be represented by the underlying PLI as follows:

$$\mathsf{AFF}_{\Delta\delta} = \{P(\varphi) \mid \varphi \in \Sigma_{\Delta\delta}\}.$$

Complications. From the analysis above it follows that when confidence increases, $\mathsf{IncMiner}_\delta$ is a continuation of BatchMiner. It only needs the mined rules from prior iteration to continue the mining.

However, the problem gets much more challenging when confidence decreases, where $\mathsf{IncMiner}_\delta$ needs to examine predecessors of the mined REEs, by reverting the mining process of BatchMiner. Intuitively, some REEs examined in the upper layers of the search lattice but deemed insufficient confidence now become sufficient under the new confidence threshold δ'. $\mathsf{IncMiner}_\delta$ has to traverse back up the search lattice to recover such newly valid REEs.

Traversing up the search lattice is tricky because the confidence of REEs exhibits no monotonicity *w.r.t.* the lattice layer (adding and removing predicates from preconditions). This means that without additional information, in order to recover the minimal valid REEs, one has to examine all predecessors of all REEs in $\Sigma_{(\sigma,\delta)}$ and $\Sigma_{<\sigma}$ (*i.e.,* REEs pruned due to insufficient support), ending up with the same complexity as rerunning BatchMiner.

Example 4: Consider an example Person relation in Table 1, which consists of five tuples (t_1-t_5), and a candidate rule φ_a : person(t_1) $\wedge$ person(t_2) $\wedge$ t_1.citizen $=$ t_2.citizen $\longrightarrow$ t_1.country $=$ t_2.country, where $\mathsf{conf}(\varphi_a, \mathcal{D}) = \frac{|\mathsf{spset}(\varphi_a,\mathcal{D})|}{|\mathsf{spset}(X,\mathcal{D})|} = \frac{3}{6} = 0.5$. Adding a predicate t_1.area-code $=$ t_2.area-code to precondition X in φ_a yields a new rule φ_b : person(t_1) $\wedge$ person(t_2) $\wedge$ t_1.citizen $=$

t_2.citizen $\wedge$ t_1.area-code $= t_2$.area-code $\rightarrow$ t_1.country $= t_2$.country, with increased confidence $\text{conf}(\varphi_b, \mathcal{D}) = \frac{|\text{spset}(\varphi_b,\mathcal{D})|}{|\text{spset}(X,\mathcal{D})|} = \frac{3}{3} = 1.0$. However, adding another predicate t_1.city $= t_2$.city to X in φ_a yields another rule φ_c : person(t_1) $\wedge$ person(t_2) $\wedge$ t_1.citizen $= t_2$.citizen $\wedge$ t_1.city $=$ t_2.city $\rightarrow$ t_1.country $= t_2$.country, with lower confidence $\text{conf}(\varphi_c, \mathcal{D}) = \frac{|\text{spset}(\varphi_c,\mathcal{D})|}{|\text{spset}(X,\mathcal{D})|} = \frac{0}{1} = 0$. These demonstrate that adding different predicates to a rule can either increase or decrease its confidence, thus showing that confidence exhibits no monotonicity *w.r.t.* predicate addition. □

Summarizing search lattice via sampling. To cope with the new challenges, we use new data structures. To avoid recomputing confidence for all REEs in the upper layers of the search lattice, the key is to summarize some information of the search lattice such that only relevant REEs are examined and reevaluated for confidence. Storing the entire search lattice is costly, taking $O(2^{|P|})$ space, where $|P|$ is the number of all predicates. Instead, we sample representative nodes in the search lattice to reduce storage overhead, while maintaining the ability to recover the full lattice. This ability is crucial for ensuring the correctness of IncMiner$_\delta$, *i.e.*, it can recover all REEs mined by BatchMiner, and is maintained by introducing the coverage constraint in the sampling problem defined below.

Intuitively, although the confidence of REEs has no monotonicity *w.r.t.* predicate addition, it still exhibits some degree of continuity. That is, the difference between confidence of similar REEs tends to be small. Such proximity enables succinct summarization of the search lattice by grouping similar REEs together.

Figure 4 depicts the summarization of a search lattice. The dashed circle indicates for a sampled REE φ, the set of nearby REEs that can be recovered from φ. The neighboring nodes can be recovered by enumerating REEs that share the same K predicates with φ, where K is a parameter to control the coverage of the sample. Figure 5 illustrates the coverage of a sample with varying K. When $K = 1$, the sample covers REEs at the same level with common predecessor pred1. The coverage increases as K increases: when $K = 2$, it covers the ones with common predecessors either pred1 or pred2.

More specifically, each sampled node is associated with three elements $(\varphi, K, \text{maxConf})$, where φ is the sampled REE, K is the sample coverage radius as defined above, and maxConf is the maximum confidence of REEs covered by the sample.

The sampling speeds up the process as each sampled node summarizes a group of nodes with a maximum confidence (maxConf). If confidence threshold $\delta >$ maxConf, then the sampled group can be skipped altogether, saving the enumeration and confidence calculation, the most costly part since it needs to scan the underlying dataset. For instance, consider the AFF region enclosed by two horizontal dashed lines in Figure 4: samples further above the AFF region should have a confidence range below δ' (samples near AFF border may cover some REEs in AFF). Thus, they can be skipped during incremental mining, eliminating redundant computation.

Sampling as an optimization problem. Denote by $\mathcal{L}$ the search lattice. The sampling problem is to find a subset $\mathcal{P} \subseteq \mathcal{L}$, and the associated radius K_p for each $p \in \mathcal{P}$, such that the following constraints are satisfied: (a) *coverage constraint* to ensure that all nodes in the lattice $\mathcal{L}$ are covered by at least one sampled node:

$$\forall v \in \mathcal{L},\ \exists p \in \mathcal{P},\ v \in N(p, K_p),$$

where $N(p, K_p)$ denotes the set of nodes covered by the sampled node p under radius K_p, and (b) *storage budget constraint* that limits the maximum number of sampled nodes: $|\mathcal{P}| \le B$.

The objective is to minimize the average radius of sampled nodes: $\frac{1}{|P|}\sum_{p\in\mathcal{P}} K_p$, which minimizes the computation required to evaluate the confidence of neighboring nodes of each sampled node.

```
Procedure IncExpand_δ
Input: $\mathcal{D}_s, Q_0, p_0, \delta, \sigma$, the current set $\Sigma$ of $\lambda$-bounded REEs, and the set $S$ of sampled lattice nodes.
Output: An updated set $\Sigma$ of minimal REEs and updated lattice sample $S'$.
1. $S' := S$;
  ... // same as lines 8-13 in Figure 1.
2.     if supp($\varphi$) $\geq \sigma$ then
3.         if $\varphi$ is not covered by $S'$ then
4.             $S' := S' \cup \{(\varphi, K, [\mathsf{conf}(\varphi), \mathsf{conf}(\varphi)])\}$;
5.         else
6.             Pick the first element $s$ in $S'$ such that $s$ covers $\varphi$;
7.             $m' = \max(s.\mathsf{maxConf}, \mathsf{conf}(\varphi))$;
8.             $S' := S' \setminus \{s\} \cup \{(s.\varphi, s.K, m')\}$;
               ... // update Q as in lines 15-16 in Figure 1.
9. return $(\Delta\Sigma, S')$;
```

Fig. 6. Subroutine IncExpand$_\delta$ with greedy heuristic

Proposition 1: *The decision version of the optimization problem of sampling the search lattice, as formulated above, is* NP*-complete.* □

Proof sketch: The upper bound is immediate. The lower bound is verified by reduction from Set Cover, which is NP-complete [47]. Set Cover is to decide, given a universe $U = \{u_1, u_2, \ldots, u_n\}$ of n elements, a collection of subsets $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ such that $\bigcup_{i=1}^{m} S_i = U$, and a budget k, whether there exists a sub-collection $\mathcal{S}' \subseteq \mathcal{S}$ with $|\mathcal{S}'| \leq k$ that covers U. Given an instance of Set Cover, we construct a lattice $\mathcal{L}$, a storage budget B, a coverage radius $K_p = 1$ for each node $p \in \mathcal{L}$. We show that there exists a cover $\mathcal{S}' \subseteq \mathcal{S}$ with $|\mathcal{S}'| \leq k$ that covers U for Set Cover if and only if there exists a set of sampled nodes $\mathcal{P} \subseteq \mathcal{L}$ that satisfies the coverage and budget constraints. The full proof is in Appendix A. □

Approximation with greedy heuristic. In light of the intractability, we develop a heuristic sampling procedure in Figure 6, which is embedded in the IncExpand$_\delta$ subroutine. This procedure reuses most parts of the Expand subroutine in Figure 1, but it maintains the set of sampled lattice nodes using a greedy heuristic.

For each examined REE φ, if it is not yet covered by any samples in S', it is constructed into a new sample (line 4). This ensures that IncMiner$_\delta$ can reconstruct the full search lattice by enumerating neighboring REEs covered by all samples, a key property for establishing IncMiner$_\delta$'s completeness (see Proposition 2). The radius parameter K is fixed during runtime to further reduce the computation overhead (see Section 7). Otherwise, we pick the first sample s in S such that s covers φ, and update maximum confidence of sample s. Although φ can be covered by multiple samples, updating the confidence range of one covering sample suffices to guarantee that φ will be discovered when it becomes valid as δ decreases.

Example 5: Consider an REE φ with conf $= 0.8$, and a covering sample s with confRange $= [0.5, 0.6]$. We update s's confidence range to $[0.5, 0.8]$. When δ decreases to $\delta' = 0.7$, φ becomes valid. Since s's updated confidence range covers δ', all neighbors of s will be enumerated for newly valid REEs, including φ. Note that this discovery process of φ does not rely on other covering samples. □

5.2 Incremental algorithms in Response to $\Delta\delta$

We first develop IncMiner$_\delta$ with completeness guarantee, *i.e.,* a complete cover of $\Sigma_{(\sigma,\delta)}$ is mined. We then give an approximate algorithm to improve efficiency while providing accuracy guarantee.

As shown in Figure 7, IncMiner$_\delta$ utilizes the sampled lattice S. Similar to IncMiner$_\sigma$, it also

```
Input: Schema $\mathcal{R}$, dataset $\mathcal{D}$, $\lambda = (\sigma, \delta)$, the set $\Sigma$ of $\lambda$-bounded REEs, search lattice samples $S$, $\Delta\delta \in [-1, 1]$
    such that $\delta + \Delta\delta \in [0, 1]$.
Output: $\Delta\Sigma_+, \Delta\Sigma_-$, and the updated search lattice samples $S'$.
1.  $\Delta\Sigma_+ := \emptyset$;  $\Delta\Sigma_- := \emptyset$;  $S' := S$;
2.  if ($\Delta\delta > 0$) then
3.     $\Delta\Sigma_- := \{\varphi \in \Sigma \mid \mathsf{conf}(\varphi) < \delta + \Delta\delta\}$;
4.     for each $p_0 \in$ RHS do
5.        $Q := \{\varphi \in \Delta\Sigma_- \mid \varphi.p_0 = p_0\}$;
6.        $(\Delta\Sigma_+, S') := \mathsf{IncExpand}_\delta(\mathcal{D}, Q, p_0, \delta + \Delta\delta, \sigma, \Delta\Sigma_+, S')$;
7.     return $(\Delta\Sigma_+, \Delta\Sigma_-, S')$;
8.  else // $\Delta\delta < 0$
9.      for each $s \in S$ do // Enumerate $S$ by size in descending order
10.            if $s$.confRange.max $\geq \delta + \Delta\delta$ then
11.                $\Delta\Sigma_+ := \Delta\Sigma_+ \cup \{\varphi \in \mathsf{Neighbor}(s) \mid \mathsf{conf}(\varphi) \geq \delta + \Delta\delta\}$;
12. return $(\Delta\Sigma_+, \emptyset, S')$;
```

Fig. 7. Algorithm $\mathsf{IncMiner}_\delta$

separates two cases.

(1) When $\Delta\delta > 0$, REEs in Σ with insufficient confidence are put into $\Delta\Sigma_-$ for removal (line 3). For each predicate p_0 in RHS, the relevant REEs in $\Delta\Sigma_-$ are put back into the search queue Q for further expansion (lines 5-6), using $\mathsf{IncExpand}_\delta$ given in Figure 6.

(2) When $\Delta\delta < 0$, $\mathsf{IncMiner}_\delta$ traverses the search lattice from bottom to top (line 9). The rationale is that when $\Delta\delta$ is small, the newly valid REEs should be close to those minded REEs of Σ in the search lattice. More specifically, for each sample s in S, if its confidence range covers the new threshold $\delta + \Delta\delta$, then its neighbors are enumerated for newly valid REEs (lines 10-11).

Example 6: Continuing with Example 2, the data analyst now decides to lower the confidence threshold δ from 0.9 to 0.8. As shown in Figure 4, when δ decreases, $\mathsf{IncMiner}_\delta$ examines all the sampled REEs whose confidence range covers the new threshold 0.8. One such sample, φ_s, is highlighted by a dashed circle Figure 4, with a confidence range between 0.7 and 0.9. Suppose $\varphi_s = \text{person}(t) \wedge t.\text{status} = \text{“working”} \wedge t.\text{country} = \text{“Norway”} \rightarrow t.\text{age} \leq 66$.

By enumerating φ_s's neighbors with radius 1, *i.e.,* substituting one of the predicates in φ_s's precondition, we find $\varphi_6 = \text{person}(t) \wedge t.\text{status} = \text{“working”} \wedge t.\text{country} = \text{“UK”} \rightarrow t.\text{age} \leq 66$. This rule indicates that if a person is working and resides in the UK, then s/he is likely to be under 66, which is the State Pension age in the UK. Other relevant REE samples are examined in the same way. □

Recall that in the sampling scheme, all sampled nodes collectively cover the full lattice, which implies the completeness of $\mathsf{IncMiner}_\delta$, as stated in Proposition 2. In other words, $\mathsf{IncMiner}_\delta$ guarantees to discover all newly valid REEs on confidence update.

Proposition 2: *Denote by Σ_s the set of sampled vertices that cover the search lattice under threshold σ and δ, and by δ' the decreased confidence. Then for every minimal (σ, δ')-bounded* REE *φ, there exists a sample in Σ_s that covers φ.* □

Proof sketch. (1) By BatchMiner and $\mathsf{IncMiner}_\delta$, every σ-frequent REE is either directly traversed by $\mathsf{IncExpand}_\delta$, or has a predecessor in the search lattice that is (σ, δ)-bounded and is traversed. (2) For every minimal (σ, δ')-bounded REE φ, by the definition of minimality, and that $\delta' < \delta$, none of its predecessor can be (σ, δ)-bounded; therefore it is directly traversed by $\mathsf{IncExpand}_\delta$. (3) For every traversed node, $\mathsf{IncExpand}_\delta$ ensures that it is either covered by an existing sample within a predefined radius K, or becomes a new sample itself. (4) Thus, each minimal (σ, δ')-bounded

```
Input: Same input as IncMiner$_\delta$ in Figure 7, and recall $\beta$.
Output: Same output as IncMiner$_\delta$.
    ... // same as lines 1-7 in Figure 7.
8.  else // $\Delta\delta < 0$
9.      $N := \Sigma_{s \in S}$ $[1 - s.CDF(\delta + \Delta\delta)] \times s.N$; // Num. of valid REEs.
10.     FN := 0;  FN$_{\max}$ := $N \times (1 - \beta)$;  $S_\approx := \emptyset$;
11.     for each $s \in S$ do
12.         $n := [1 - s.\mathrm{CDF}(\delta + \Delta\delta)] \times s.N$ ;
13.         if FN + $n \leq$ FN$_{\max}$ then
14.             FN := FN + $n$;
15.         else $S_\approx := S_\approx \cup \{s\}$;
16.     for each $s \in S_\approx$ do // Enumerate $S_\approx$ by size in descending order.
            ... // Same as lines 10-11 in Figure 7.
17.     $\Delta\Sigma_+ := \{\varphi \in \Delta\Sigma_+ \mid \mathrm{minimize}(\varphi)\}$;
18. return $(\Delta\Sigma_+, \emptyset, S)$
```

Fig. 8. IncMiner$^{\approx}_{\delta}$ when $\Delta\delta < 0$ with recall guarantee.

REEs is covered by at least one sample in Σ_s, proving the proposition. □

Remarks. Proposition 2 relies on two properties of the batch mining algorithm: (1) it is deterministic; and (2) it does not approximate the mining results, *i.e.,* the use of optimizing heuristics that prematurely prune some parts of the search lattice rendering incomplete mining results *w.r.t.* support and confidence requirements. BatchMiner satisfies both, whereas other variants of REE mining (*e.g.,* [24, 28, 29]) do not. Ensuring the completeness while incrementalizing these variants remains an important direction for future work.

Algorithm IncMiner$^{\approx}_{\delta}$. We show how to relax IncMiner$_\delta$ to further improve efficiency, while guaranteeing recall bound β, *i.e.,* at least β% of REEs mined by the complete algorithm are discovered in the probabilistic version. The main overhead of IncMiner$_\delta$ is that, when $\Delta\delta < 0$, it has to recompute confidence for a lot of samples and their neighboring REEs. To reduce the cost, the neighbor information of each sampled REE need to be summarized more precisely.

To speed it up, for each sampled REE, instead of confRange, we keep a CDF (*cumulative distribution function*) of the confidences of the neighboring REEs, where CDF$(\delta') = Pr.(\text{conf} \leq \delta')$, *i.e.,* the fraction of neighboring REEs whose confidence is below δ'. Intuitively, given a sampled vertex s, if only few neighboring REEs have sufficient confidence, *i.e.,* $Pr.(\text{conf} \leq \delta') \lesssim 1$, we can skip enumerating neighbors of s without losing too many valid REEs.

However, the reverse is not true. When $Pr(\text{conf} \leq \delta') \gtrsim 0$, *i.e.,* most accounted neighboring REEs have high confidence, we cannot simply add all neighbors of s into $\Delta\Sigma_+$. Recall from Figure 6 that for sampling efficiency, the CDF of each sample REE only represents a subset of neighboring REEs. Therefore, directly adding all neighbnors can lead to unpredictable and high false positives.

To ensure that the approximation does not break recall guarantee, we keep FN for the number of forgone valid REEs. Denote by N the number of valid REEs; then recall can be expressed as recall $= 1 - \frac{N - \text{FN}}{N}$. Substituting these for constraints recall $\geq \beta$, we maintain FN $\leq \beta \times N$, an invariant in incremental mining.

Given these, we develop algorithm IncMiner$^{\approx}_{\delta}$ in Figure 8. It first computes the number of all valid REEs under new confidence threshold $\delta + \Delta\delta$ by summing up the number of valid REEs covered by each sample (line 9). It then initializes the counter FN for the number of forgone valid REEs, and derives the upper bound of FN as FN$_{\max}$ $= N \times \beta$, to maintain the recall guarantee (line 10).

It then extracts the subset $S_\approx$ of samples s in S for examination, such that the recall is guaranteed (lines 11-15). For each sample s in $S_\approx$, it follows the same enumeration procedure as IncMiner$_\delta$

(lines 10-11 in Figure 7) to update the set $\Delta\Sigma_+$ of newly valid REEs.

Finally, it minimizes each REE in $\Delta\Sigma_+$ to remove redundant predicates. A non-minimal REE φ can be introduced when its minimal predecessor $\varphi_{\min}$ is dropped by the approximation. The mining algorithm finds such $\varphi_{\min}$ in $\Delta\Sigma_+$ to prove the non-minimality of φ.

Example 7: Consider a sample φ_s with $Pr.(\text{conf} \leq \delta') = 0.9$, and neighbor size $s.N = 100$. Discarding all neighbors of s will increase false negative FN by $100 \times (1 - 0.9) = 10$. □

Relative boundedness. As shown in Figure 4, during the incremental mining process for $\Delta\delta < 0$, only samples within AFF and those along the AFF border are reexamined. As these samples also cover newly valid REEs, their confidence ranges cover δ'. The number of samples along the AFF border is $O\left(\frac{|\text{AFF}|}{|S|}\right)$, where $|S|$ denotes the average number of REEs covered by a sample. Consequently, the number of examined REEs outside of AFF is $O(|\text{AFF}|)$, and the overall number of reexamined REEs is linear in the size of AFF, *i.e.,* $O(|\text{AFF}|)$. Thus IncMiner_δ is relatively bounded to BatchMiner. Similarly, $\text{IncMiner}_\delta^{\approx}$ is also relatively bounded to BatchMiner.

6 Incremental Algorithm For $\Delta\lambda$

Putting the two incremental algorithms in Sections 4 and 5 together, this section develops an incremental REE discovery algorithm in response to updates to parameter λ (Section 6.1). We also parallelize the algorithm, and show that it is both bounded and parallelly scalable relative to batch algorithm BatchMiner (Section 6.2).

6.1 Incremental Algorithm in Response to $\Delta\lambda$

The incremental algorithm in response to $\Delta\lambda = (\Delta\sigma, \Delta\delta)$ is shown in Figure 9, referred to IncMiner_λ. It separates the following cases.

(1) When both $\Delta\sigma$ and $\Delta\delta$ are positive, IncMiner_λ filters unqualified REEs in Σ and adds them to the set $\Delta\Sigma_-$. It then continues expanding REEs in $\Delta\Sigma_-$ using procedure IncExpand_λ (lines 1-3), such that the expanded REEs have confidence above $\delta + \Delta\delta$ and at the same time, do not decrease support below $\sigma + \Delta\sigma$. IncExpand_λ has the same logic of IncExpand_σ in Figure 3 and IncExpand_δ in Figure 7, maintaining both auxiliary states $\Sigma_{<\sigma'}$ and S' simultaneously.

(2) When $\Delta\sigma > 0$ and $\Delta\delta < 0$, IncMiner_λ filters REEs in Σ according to the updated parameters (lines 4-6), and then discovers lower confidence REEs by traversing the search lattice just like IncMiner_δ.

(3) When $\Delta\sigma < 0$ and $\Delta\delta > 0$, it first filters unqualified REEs, and then continues mining using IncExpand_λ, along the same lines as the corresponding cases in IncMiner_σ and IncMiner_δ.

(4) When both $\Delta\sigma$ and $\Delta\delta$ are negative, IncMiner_λ first continues mining for lower support using IncExpand_λ, and then traverses back the search lattice following the same procedure as IncMiner_δ (Figure 7, lines 9-11), to find REEs with decreased confidence.

Example 8: Continuing with Examples 3 and 6, the user decides to lower support (σ from 100 to 50) and confidence (δ from 0.9 to 0.8) simultaneously. Let $F = (\Sigma, \Sigma_{<\sigma}, S)$ be the set of auxiliary states after mining with the old σ and δ. IncMiner_λ first continues mining for low support REEs from the current states F (Figure 2). It differs from IncMiner_σ in that REEs with lower confidence are considered valid, *e.g.,* REEs with support 80 and confidence 0.85.

Next, IncMiner_λ traverses back the search lattice from states F, to uncover low confidence REEs neglected in prior mining (Figure 4), *e.g.,* φ_6 (Example 6) is one of such uncovered REEs. □

Relative boundedness. Following the analyses for IncMiner_σ and IncMiner_δ, one can verify that each of these four cases is bounded relative to the batch algorithm BatchMiner; hence so is IncMiner_λ.

```
Input: $\mathcal{R}, \mathcal{D}, \lambda = (\sigma, \delta), \Sigma, \Delta\sigma$, and $\Delta\delta \in [-1, 1]$ as in Figures 3 and 7, and auxiliary states $\Sigma_{<\sigma}$ and $S$ as in
    those algorithms.
Output: The set $\Delta\Sigma_+, \Delta\Sigma_-, \Sigma_{<\sigma'}$, and $S'$ as above.
1.  if $\Delta\sigma > 0 \wedge \Delta\delta > 0$ then
2.      $\Delta\Sigma_- := \{\varphi \in \Sigma \mid \mathsf{supp}(\varphi) < \sigma + \Delta\sigma \vee \mathsf{conf}(\varphi) < \delta + \Delta\delta\}$;
3.      $(\Delta\Sigma_+, \Sigma_{<\sigma'}, S') := \mathsf{IncExpand}_\lambda(\mathcal{D}, \Delta\Sigma_-, \sigma + \Delta\sigma, \delta + \Delta\delta, \Sigma_{<\sigma}, S)$;
4.  elseif $\Delta\sigma > 0 \wedge \Delta\delta < 0$ then
5.      $\Delta\Sigma_- := \{\varphi \in \Sigma \mid \mathsf{supp}(\varphi) < \sigma + \Delta\sigma\}$;
6.      ... // Update $\Delta\Sigma_+$ following lines 9-11 in Figure 7;
7.  elseif $\Delta\sigma < 0 \wedge \Delta\delta > 0$
8.      $\Delta\Sigma_- := \{\varphi \in \Sigma \mid \mathsf{conf}(\varphi) < \delta + \Delta\delta\}$;
9.      $(\Delta\Sigma_+, \Sigma_{<\sigma'}, S') := \mathsf{IncExpand}_\lambda(\mathcal{D}, \Delta\Sigma_-, \sigma + \Delta\sigma, \delta + \Delta\delta, \Sigma_{<\sigma}, S)$;
10. else  // $\Delta\sigma < 0 \wedge \Delta\sigma < 0$
11.     $Q := \{\varphi \in \Sigma_{<\sigma} \mid \mathsf{supp}(\varphi) \geq \sigma + \Delta\sigma\}$;
12.     $(\Delta\Sigma_+, \Sigma_{<\sigma'}, S') := \mathsf{IncExpand}_\lambda(\mathcal{D}, Q, \sigma + \Delta\sigma, \delta + \Delta\delta, \Sigma_{<\sigma}, S)$;
13.     ... // Update $\Delta\Sigma_+$ following lines 9-11 in Figure 7;
14. return $(\Delta\Sigma_+, \Delta\Sigma_-, \Sigma_{<\sigma'}, S')$;

Procedure $\mathsf{IncExpand}_\lambda$
Input: $\mathcal{D}_s, Q_0, \delta, \sigma$, the current set $\Sigma_{<\sigma}$ of pruned REEs, and the set $S$ of sampled lattice nodes.
Output: $\Delta\Sigma_+, \Sigma'_{<\sigma}, S'$ as above.
15. $\Sigma'_{<\sigma} := \Sigma_{<\sigma}$;  $S' := S$; $\Delta\Sigma_+ = \emptyset$;
16. for each $p_0 \in$ RHS do
17.     $Q := \{\varphi \in \Sigma_{<\sigma} \mid \varphi.p_0 = p_0\}$;
18.     while $Q \neq \emptyset$ do
19.         // Update $\Delta\Sigma_+$ and $\Sigma'_{<\sigma}$ following IncExpand in Figure 3;
20.         // Update $S'$ following lines 2-8 in Figure 6;
21. return $(\Delta\Sigma_+, \Sigma'_{<\sigma}, S')$;
```

Fig. 9. Algorithm $\mathsf{IncMiner}_\lambda$

6.2 Parallel Rule Discovery

To scale with large datasets, we parallelize $\mathsf{IncMiner}_\lambda$, denoted by $\mathsf{PIncMiner}_\lambda$. We show that $\mathsf{PIncMiner}_\lambda$ is parallelly scalable, *i.e.,* it can scale with large datasets by adding more processors.

Parallel scalability. We adapt the notion of [51] to characterize the effectiveness of parallel algorithms. Consider a sequential algorithm $\mathcal{I}$ for the incremental REE discovery problem. Let $t(|\mathcal{D}|, \Delta\lambda, \lambda)$ be the worst-case runtime of $\mathcal{I}$ when solving the instance $(\mathcal{D}, \Delta\lambda, \lambda)$ of the incremental discovery problem. We say that a parallel algorithm $\mathcal{I}_p$ is *parallelly scalable relative to* $\mathcal{I}$ if on any instance $(\mathcal{D}, \Delta\lambda, \lambda)$ of the problem, the runtime of $\mathcal{I}_p$ using n processors in response to parameter updates $\Delta\lambda$ can be expressed as:

$$T(|\mathcal{D}|, \Delta\lambda, \lambda, n) = O\big(\frac{t(|\mathcal{D}|, \Delta\lambda, \lambda)}{n}\big).$$

Intuitively, the parallel scalability guarantees speedup of parallel algorithm $\mathcal{I}_p$ relative to a "yardstick" sequential algorithm $\mathcal{I}$. Such $\mathcal{I}_p$ can reduce the cost of $\mathcal{I}$ when more processors are used.

Parallelization. The batch algorithm BatchMiner has been parallelized in [28] under Bulk Synchronous Parallel (BSP) [85] model and shown parallelly scalable. Given n machines, a designated coordinator partitions the discovery job into $n-1$ work units, distributes the work units to workers, and synchronizes the computation. Each worker is responsible for its allocated work units for rule discovery. The workload is dynamically balanced among workers to ensure efficient processing and thus, the parallel scalability.

What makes BatchMiner amenable to parallelism is the levelwise expansion strategy (Expand in

Table 2. Dataset characteristics

Name	Type	#tuples	#attributes	#relations
Adult [50, 55, 63]	real-life	32,561	15	1
Airport [55, 63]	real-life	55,113	18	1
Hospital [10, 21, 55]	real-life	114,919	15	1
Inspection [55, 68]	real-life	220,940	17	1
NCVoter [55, 63]	real-life	1,681,617	12	1
DBLP [80]	real-life	1,799,559	18	3
Parksong [1, 9]	real-life	5,002,872	27	1

Figure 1), where each worker expands a subset of candidate REEs in parallel and aggregate results at the end. Since IncMiner$_\lambda$ adopts the same expansion strategy (IncExpand in Figure 3), it can also be paralleliszed using BSP. It differs from the parallel batch counterpart only in the following.

(1) Maintenance of auxiliary data structures. Since such structures are employed and updated in the expanding procedure of each candidate REE, they can be maintained within each work unit without coordination, maintaining the same degree of parallelism.

(2) Lattice traversal via samples in IncMiner$_\lambda$ (*i.e.,* IncMiner$_\delta$ in Figure 7 and IncMiner$_\delta^{\approx}$ in Figure 8). The traversal of sample neighbors adopts a levelwise enumeration strategy just like BatchMiner, and hence can be partitioned into work units and parallelized under the BSP model in the same way as BatchMiner.

Following the analysis of the parallelized BatchMiner [28], one can conclude that PIncMiner$_\lambda$ is *parallelly scalable relative to* IncMiner$_\lambda$. Along the same lines, we also parallelize algorithms IncMiner$_\sigma$ and IncMiner$_\delta$, both with the parallel scalability.

7 Experimental Study

Using real-life data, this section experimentally evaluated (a) the efficiency and (b) (parallel) scalability of the incremental algorithms in response to parameter updates, and (c) the effectiveness of the optimization strategies. We also provide (d) guidelines for new discovery paradigms based on the incremental algorithms, and (e) a test on the quality of rules discovered in practice.

Experimental setting. We start with the experimental setting.

Datasets. We used seven real-world datasets $\mathcal{D}$ from prior studies, as summarized in Table 2. We discovered rules from the entire $\mathcal{D}$.

Baselines. We implemented PIncMiner$_\lambda$ in Go and evaluated it against the following. *(1) Batch baselines*: BatchMiner [28], the SOTA REE mining algorithm (Section 2), and DCFinder [10], the SOTA method for discovering DCs. *(2) Incremental baseline:* IApriori [7], which mines association rules upon parameter updates; its rules are defined on a single tuple with constant predicates only, a special case of REEs. We implemented IApriori using the Spark FP-Growth library [77]. *(3) Variants of* PIncMiner$_\lambda$: IncMiner$_{\delta(0.7)}^{\approx}$ is IncMiner$_\delta^{\approx}$ with minimum recall 0.7; IncMiner$_{\lambda_{\text{NS}}}$ is PIncMiner$_\lambda$ without sampling the search lattice (Section 5), and traverses the entire lattice again when $\Delta\delta < 0$ since confidence exhibits no anti-monotonicity. We parallelized the baselines for a fair comparison.

Default parameters. By default, we set the number of machines $n = 20$, the support threshold $\sigma = 10^{-6} \cdot |\mathcal{D}|^2$, the confidence threshold $\delta = 0.75$, the radius $K = 3$ for sampling (Section 5), the bounds for recall $\beta = 0.7$, $\Delta\sigma = \times 10^{\pm 1}$, and $\Delta\delta = \pm 0.1$.

Configuration. We conducted experiments on a cluster of up to 21 virtual machines, each powered by 8GB RAM and a 2.20 GHz core. We do not include the time for loading datasets and precomputing auxiliary data structures like PLI. Since batch mining takes long, we set a timeout threshold at 10 hours. Following BatchMiner [28] (Section 2), we target REEs pertaining to an application of users' interest *w.r.t.* RHS and P_0. We mine only bi-variable REEs for a consistent comparison with DCFinder, which mines bi-variable DCs.

Experimental results. We next report our findings. For the lack of space we show results on some datasets; the others are consistent.

Exp-1: Efficiency. We tested the effectiveness of incremental rule discovery in response to updates to support $\Delta\sigma$, confidence $\Delta\delta$ and both $\Delta\lambda = (\Delta\sigma, \Delta\delta)$, when $\Delta\sigma$ and $\Delta\delta$ are positive or negative.

Varying $\Delta\sigma$ ($\Delta\sigma > 0$). With initial $\sigma = 10^{-6}|\mathcal{D}|^2$, we varied $\Delta\sigma$ such that updated $\sigma' = \sigma \oplus \Delta\sigma$ is from $10^{-5}|\mathcal{D}|^2$ to $10^{-2}|\mathcal{D}|^2$. As shown in Figure 10(a) on Adult, (1) PIncMiner$_\lambda$, IncMiner$^{\approx}_{\delta(0.7)}$ and IncMiner$_{\lambda_{NS}}$ outperform BatchMiner and DCFinder by 468× and 1314× on average, up to 658× and 1956×, respectively. Their runtime is under 1s since they only filter previously mined rules without further mining. (2) Batch algorithms get faster when σ increases since by the anti-monotonicity of support, there are less λ-bounded rules. In contrast, the three incremental ones are insensitive to $\Delta\sigma$. (3) IApriori takes less than 1s, since it mines single-tuple rules with only constant predicates, which are much simpler than REEs defined on multiple tuples with variable predicates. It exhibits similar performance across all subsequent experiments. (4) PIncMiner$_\lambda$, IncMiner$^{\approx}_{\delta(0.7)}$ and IncMiner$_{\lambda_{NS}}$ perform comparably as expected, since they only differ when $\Delta\delta < 0$.

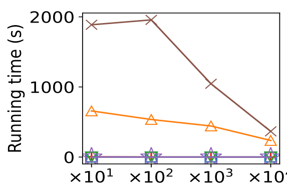
Varying $\Delta\sigma$ ($\Delta\sigma < 0$). Starting with $\sigma = 10^{-2}|\mathcal{D}|^2$, we varied $\Delta\sigma$ such that updated $\sigma' = \sigma \oplus \Delta\sigma$ is from $10^{-3}|\mathcal{D}|^2$ to $10^{-6}|\mathcal{D}|^2$. As shown in Figure 10(b) on Inspection, (1) all algorithms but IApriori take longer when $\Delta\sigma$ decreases. This is because for batch algorithms, smaller σ means more λ-bounded rules to mine. For incremental ones, when $\Delta\sigma < 0$, they continue the search down the lattice; so the larger $\Delta\sigma$ is, the longer it takes to enumerate all rules for the reduced σ'. (2) The three incremental algorithms perform similarly, beating BatchMiner by 6× on average, up to 15×. (3) PIncMiner$_\lambda$ and IncMiner$^{\approx}_{\delta(0.7)}$ perform similarly to IncMiner$_{\lambda_{NS}}$, indicating that sampling overhead is negligible. (4) DCFinder takes 5h with $\sigma \geq 10^{-5}|D|^2$ and times out with smaller σ as it does not scale with large number of predicates. We do not report DCFinder in the subsequent experiments as it times out on other λ's for the same reason.

Varying $\Delta\delta$ ($\Delta\delta > 0$). From initial $\delta = 0.7$, we varied $\Delta\delta$ such that updated $\delta' = \delta + \Delta\delta$ is from 0.7 to 0.95. As shown in Figure 10(c) on Inspection, (1) PIncMiner$_\lambda$, IncMiner$^{\approx}_{\delta(0.7)}$ and IncMiner$_{\lambda_{NS}}$ beat BatchMiner by 31× on average, up to 92×. Unlike the case of $\Delta\sigma > 0$, search has to be conducted when $\Delta\delta > 0$ as we no longer have the anti-monotonicity of support. (2) The three incremental ones perform comparably by continuing down the search lattice, without back traversal. (3) DCFinder times out on all $\Delta\delta$ (not shown).

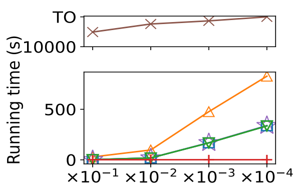
Varying $\Delta\delta$ ($\Delta\delta < 0$). Starting with initial $\delta = 0.95$, we varied $\Delta\delta$ such that updated $\delta' = \delta + \Delta\delta$ is from 0.9 to 0.7. As shown in Figure 10(d), (1) consistent with Figure 10(c) for $\Delta\delta > 0$, PIncMiner$_\lambda$ and IncMiner$^{\approx}_{\delta(0.7)}$ beat BatchMiner by 4× and 9× on average, respectively, up to 5× and 13×. (2) Both incremental and batch algorithms are less sensitive to $|\Delta\delta|$ than to $|\Delta\sigma|$ as confidence exhibits no anti-monotonicity. (3) When $\Delta\delta < 0$, sampling is very effective, as the search must traverse up the lattice (Section 5). Without sampling, IncMiner$_{\lambda_{NS}}$ takes as long as BatchMiner. (4) IncMiner$^{\approx}_{\delta(0.7)}$ is 2× faster than PIncMiner$_\lambda$ on average, up to 2.6×. The speedup comes with the price of lower recall (Figure 10(r) in Exp-3).

Varying $\Delta\lambda$. We varied $\Delta\sigma$ and $\Delta\delta$ simultaneously, such that updated σ' is from $10^{-6}|\mathcal{D}|^2$ to $10^{-2}|\mathcal{D}|^2$, and updated δ' is from 0.7 to 0.9. As shown in Figures 10(e)–10(h) for different combinations of positive and negative $\Delta\sigma$ and $\Delta\delta$, (1) in all cases, PIncMiner$_\lambda$ and IncMiner$^{\approx}_{\delta(0.7)}$ beat BatchMiner and DCFinder. (2) The trend is dominated by $\Delta\sigma$: the speedup is most significant when

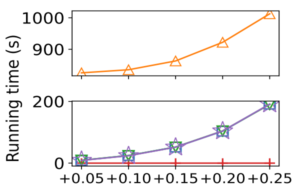
△ batch □ IncMiner ▽ IncMiner0.7 ☆ IncMinerNS + IApriori × DCFinder



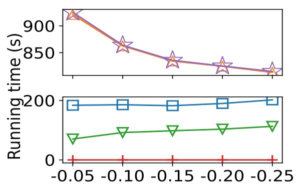
(a) [Adult]: vary $\Delta\sigma(>0)$



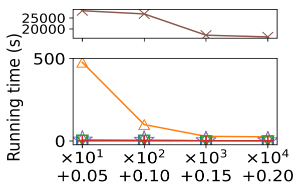
(b) [Insp.]: vary $\Delta\sigma(<0)$



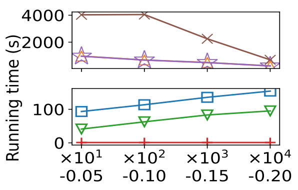
(c) [Insp.]: vary $\Delta\delta(>0)$



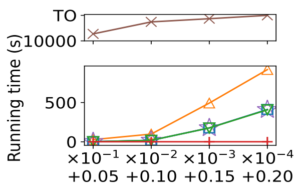
(d) [Insp.]: vary $\Delta\delta(<0)$



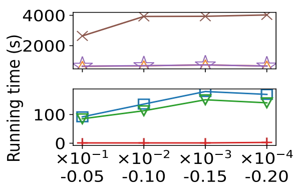
(e) [Insp.]: $(\Delta\sigma>0, \Delta\delta>0)$



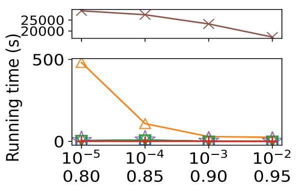
(f) [Adult]: $\Delta\lambda(\Delta\sigma>0, \Delta\delta<0)$



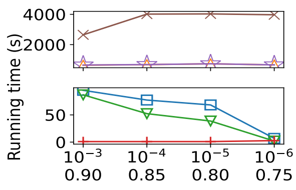
(g) [Insp.]: $\Delta\lambda(\Delta\sigma<0, \Delta\delta>0)$



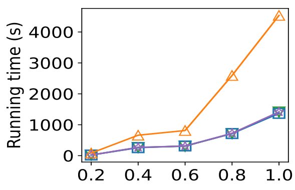
(h) [Adult]: $\Delta\lambda(\Delta\sigma<0, \Delta\delta<0)$



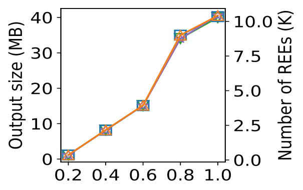
(i) [Insp.]: $\lambda(\Delta\sigma>0, \Delta\delta>0)$



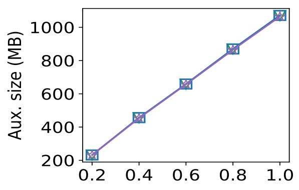
(j) [Adult]: $\lambda(\Delta\sigma<0, \Delta\delta<0)$



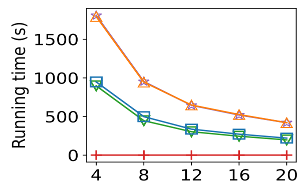
(k) [Parks.]: $|\mathcal{D}|$, runtime



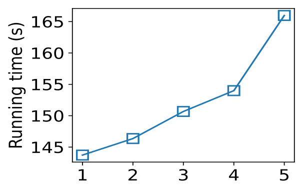
(l) [Parks.]: $|\mathcal{D}|$, output size



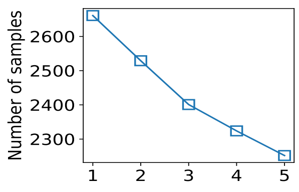
(m) [Parks.]: $|\mathcal{D}|$, aux. states



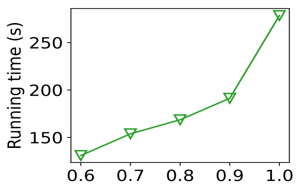
(n) [Insp.]: n, $\Delta\lambda(10^{-1}, -0.1)$



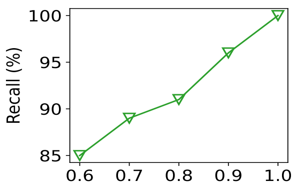
(o) [Hospital]: varying K, time



(p) [Hospital]: vary K, storage

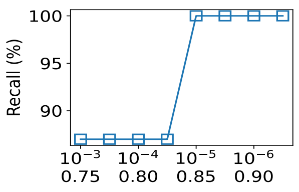


(q) [Hospital]: varying β, time

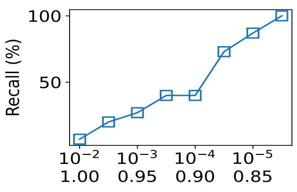


(r) [Hospital]: varying β, recall

Fig. 10. Performance evaluation (part 1 of 2).



(s) [Adult]: guideline 1, recall



(t) [Adult]: guideline 2, recall

Fig. 10. Performance evaluation (part 2 of 2).

$\Delta\sigma > 0$. Note that PIncMiner$_\lambda$ is on average 40× faster than BatchMiner in Figure 10(e) and 5× faster in Figure 10(f). The improvement decreases as $\Delta\sigma$ increases, consistent with the tests above. When $\Delta\sigma < 0$, the improvement is less sensitive to the magnitude of $\Delta\lambda$, and both take longer when $\Delta\sigma$ increases. (3) Sampling is particularly effective when $\Delta\delta < 0$, as shown in Figures 10(f) and 10(h), consistent with tests above. In this case IncMiner$_{\lambda_{\text{NS}}}$ without sampling is not much faster than BatchMiner. (4) IncMiner$_{\delta(0.7)}^{\approx}$ beats PIncMiner$_\lambda$ when $\Delta\delta < 0$ by 1.8× on average, up to 2.3×, and performs comparably when $\Delta\delta > 0$, consistent with Figures 10(c)–10(d). (5) As in the earlier tests, when $\Delta\delta > 0$, IncMiner$_{\lambda_{\text{NS}}}$ performs similarly to the other two incremental miners.

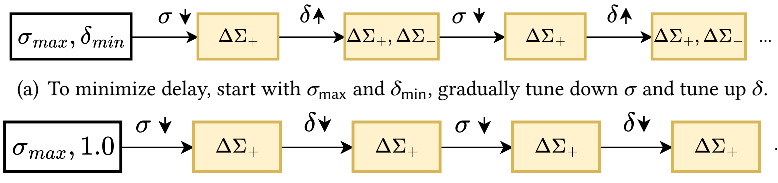
Varying λ. In addition to varying $\Delta\lambda$ with a fixed initial λ, we tested different initial λ values with fixed $\Delta\lambda$ to evaluate the effectiveness in practice, where λ continuously evolves. We simultaneously varied σ and δ such that the new σ is from $10^{-6}|\mathcal{D}|^2$ to $10^{-2}|\mathcal{D}|^2$, and δ is from 0.75 to 0.95. As shown in Figures 10(i)–10(j), (1) PIncMiner$_\lambda$ and IncMiner$_{\delta(0.7)}^{\approx}$ beat BatchMiner and DCFinder. (2) When both σ and δ increase (Figure 10(i)) the runtime of PIncMiner$_\lambda$, IncMiner$_{\delta(0.7)}^{\approx}$ and IncMiner$_{\lambda_{\text{NS}}}$ remains insensitive to the initial values, since the mining time is primarily dominated by $\Delta\sigma$, and increasing σ only needs to filter previously mined rules. (3) When both σ and δ decrease (Figure 10(j)), PIncMiner$_\lambda$ and IncMiner$_{\delta(0.7)}^{\approx}$ gets faster as search branches can be pruned earlier when mining λ-bounded rules. (4) Similar to the earlier test, IncMiner$_{\lambda_{\text{NS}}}$ performs comparably to PIncMiner$_\lambda$ and IncMiner$_{\delta(0.7)}^{\approx}$ when $\Delta\delta > 0$, but not much faster than BatchMiner when $\Delta\delta < 0$.

Exp-2: Scalability. We evaluated the scalability of the incremental algorithms by varying the size $|\mathcal{D}|$ of datasets $\mathcal{D}$ and the number n of machines employed for parallel rule discovery.

Varying $|\mathcal{D}|$. Using the largest dataset Parksong and setting $\lambda_0 = (10^{-6}|\mathcal{D}|^2, 0.6)$ and $\Delta\lambda = (\times 10^{-0.1}, +0.1)$, we evaluated the impact of $|\mathcal{D}|$ by varying the scaling factor (*i.e.,* the number of tuples) from 20% to 100%. As shown in Figures 10(k), (1) all algorithms take longer when $|\mathcal{D}|$ gets larger, as expected. (2) PIncMiner$_\lambda$ and IncMiner$_{\delta(0.7)}^{\approx}$ perform similarly, since they only differ when $\Delta\delta < 0$. (3) They outperform BatchMiner by 4× on average. Note that Parksong is an e-commerce transaction dataset in which attributes like product and shop IDs generate a large number of constant predicates. This further exacerbates the exponential growth in the number of σ-frequent REEs when support σ decreases.

We also tested on storage scalability. As shown in Figure 10(l), the output size of PIncMiner$_\lambda$, measured by the numbers of output REEs and their storage sizes, increases from 6MB to 66MB as $|\mathcal{D}|$ increases. All baselines except IncMiner$_{\delta(0.7)}^{\approx}$ show similar output sizes as they share the same λ. In Figure 10(m), auxiliary data sizes of PIncMiner$_\lambda$ (*e.g.,* PLI, pruned REEs $\Sigma_{<\sigma}$, lattice samples) increases from 232MB to 1074MB, about 58% of the input dataset size. Additionally, IncMiner$_{\lambda_{\text{NS}}}$ is only 4% smaller than PIncMiner$_\lambda$ on average, indicating that lattice samples have negligible overhead.

Varying n. Varying n from 4 to 20, with $\lambda_0 = (10^{-5}|\mathcal{D}|^2, 0.8)$, $\Delta\lambda = (\times 10^{-1}, -0.1)$, we tested the parallel scalability of the algorithms. As shown in Figure 10(n), (1) PIncMiner$_\lambda$ is 4.3× faster when



(a) To minimize delay, start with σ_{max} and δ_{min}, gradually tune down σ and tune up δ.

(b) To prioritize high quality REEs, start with σ_{max} and $\delta = 1.0$, tune down both σ and δ.

Fig. 11. Guidelines for incremental rule mining

n varies from 4 to 20. It takes 220s on Inspection of 220K tuples at $n = 20$. (2) It beats parallel BatchMiner by 1.9× on average. (3) PIncMiner$_\lambda$, IncMiner$^{\approx}_{\delta(0.7)}$ and IncMiner$_{\lambda_{\text{NS}}}$ show similarly parallelism. (4) IApriori is consistently fast as remarked earlier. (5) When $\Delta\sigma > 0$, no parallelism is needed since one node can finish in less than 1s even with the largest dataset and $\Delta\lambda$.

Exp-3: Optimization. We also evaluated the impact of configurable parameters K, β and the effectiveness of our sampling scheme on the time and space costs of incremental rule discovery. Here we set $\lambda_0 = (10^{-5}|\mathcal{D}|^2, 0.9)$ and $\Delta\lambda = (0, -0.3)$.

Varying K. We report the mining time and storage overhead of PIncMiner$_\lambda$ in Figures 10(o)–10(p) on Hospital, respectively. (1) The mining time increases with larger K, since larger radius covers more neighboring REEs, and thus incurs longer enumeration time. (2) The space cost decreases with larger K, as less sampled nodes are required to cover all REEs in the search lattice. (3) PIncMiner$_\lambda$ demonstrates a clear tradeoff between the mining time and space cost. (4) Optimal performance is observed at $K = 3$, where increasing K further does not significantly reduce the space cost, while substantially reducing mining time than larger K.

Varying β. We evaluated the mining time and recall of the discovered REEs by IncMiner$^{\approx}_{\delta}$. As shown in Figures 10(q)–10(r) on Hospital by varying the recall guarantee β from 0.6 to 1.0, (1) IncMiner$^{\approx}_{\delta(0.7)}$ takes longer as β increases due to the more thorough examination of lattice samples required for higher recall bound. Simply relaxing β from 1.0 to 0.9 largely reduces mining time since IncMiner$^{\approx}_{\delta(0.7)}$ can skip many low coverage samples. At $\beta = 0.7$, IncMiner$^{\approx}_{\delta(0.7)}$ achieves a 1.8× speedup over PIncMiner$_\lambda$, without losing much recall (true recall = 89%). (2) Its true recall is on average 15% higher than the preset lower bound β when $\beta < 1$, because samples have a large overlap in neighbors, making the false negative counter a conservative over-approximation. (3) IncMiner$^{\approx}_{\delta(0.7)}$ exhibits a clear tradeoff between time and recall, as designed.

IncMiner$_\delta$ vs. IncMiner$^{\approx}_{\delta}$. (1) As shown above, IncMiner$^{\approx}_{\delta(0.7)}$ is much faster than PIncMiner$_\lambda$ when $\Delta\delta < 0$, and performs comparably when $\Delta\delta > 0$, consistent with our design. (2) As shown in Figure 10(q), the speedup increases as the recall bound β decreases. (2) As a tradeoff, IncMiner$^{\approx}_{\delta(0.7)}$ has slightly lower recall, which is acceptable when speed is prioritized over absolute accuracy.

Exp-4: A guideline for rule discovery. As illustrated in Figure 11, incremental rule discovery suggests two strategies for tuning parameters in mining endeavors. Denote by σ_{max} the highest support of individual predicates, which is thus the highest possible σ.

(1) Small delay. Starting with $\sigma = \sigma_{\text{max}}$ and the lowest acceptable confidence δ_0, one can gradually tune down support σ while tuning up confidence δ (Figure 11(a)). This approach produces a stream of outputs with small delays between them, as every incremental mining is a natural continuation down the search lattice (Section 6) and no unnecessary recomputation is performed. Users do not have to wait idle for long periods from one output to the next, thus shortening the total time required to discover the needed rules.

(2) Prioritizing high-quality rules. If the number of mined REEs becomes a storage bottleneck, users can start with $\sigma_{\max}$ and the largest confidence $\delta' = 1.0$, and then gradually tune down both σ and δ (Figure 11(b)). This approach ensures that the highest-quality rules are discovered first. Based on the insights gained from the initial batches of mined REEs, users can abandon unhelpful rules to free up space and adjust mining criteria in future iterations [24].

Exp-5: Rule quality. To evaluate the quality of the rules mined by $\mathsf{PIncMiner}_{\lambda}$, we invited data quality experts from our industry partners to provide a set of 15 golden rules for dataset Adult. We then run $\mathsf{PIncMiner}_{\lambda}$ following the two guidelines above. As shown in Figure 11(s)-11(t), (1) in each scenario, $\mathsf{PIncMiner}_{\lambda}$ successfully discovers the complete set of golden rules. (2) It requires 5 and 8 iterations, respectively, to find the right $\lambda' = (10^{-3}, 0.8)$, and uncover the full set of golden rules under the two guidelines. (3) The end-to-end mining time is 634s and 995s, respectively. (4) Guideline 1 requires fewer iterations as its initial $\lambda_0 = (10^{-3}, 0.75)$ is closer to λ'. (5) As shown in Figure 11(s), even with mining results under the optimal λ', subsequent iterations are still required to uncover higher confidence rules. These rules would remain undiscovered at lower δ if they have λ-bounded predecessors. This highlights the need for incremental mining upon parameter updates.

We also compared the outputs of $\mathsf{PIncMiner}_{\lambda}$ and DCFinder. There are 2349 DCs mined by DCFinder, which are covered by REEs mined by $\mathsf{PIncMiner}_{\lambda}$. Additionally, under the same time constraints, $\mathsf{PIncMiner}_{\lambda}$ discovers 18% more rules that are not mined by DCFinder, since DCFinder misses certain rules with constant predicates due to its pruning strategy for speed. These findings indicate that $\mathsf{PIncMiner}_{\lambda}$ can discover rules as expressive as DCs.

Summary. We find the following. (1) Incremental rule discovery is effective. It outperforms the batch counterpart, no matter whether $\Delta\sigma$ and $\Delta\delta$ are positive or negative, up to 658×. It is 2.5× and 5× faster even when $\Delta\sigma$ and $\Delta\delta$ account for 99% and 20% of σ and δ, respectively. (2) Our sampling strategy is effective. When $\Delta\delta < 0$, $\mathsf{PIncMiner}_{\lambda}$ and $\mathsf{IncMiner}^{\approx}_{\delta(0.7)}$ beat $\mathsf{IncMiner}_{\lambda_{\mathsf{NS}}}$ by 4× and 9×, respectively. (3) Incremental rule discovery is feasible in practice. $\mathsf{PIncMiner}_{\lambda}$ takes 170s on Adult with 32K tuples when $n = 20$, when $\Delta\sigma$ and $\Delta\delta$ account for 99% and 10% of σ and δ, respectively, as opposed to 664s by the batch baseline. (4) $\mathsf{PIncMiner}_{\lambda}$ is parallelly scalable. It is 4.3× faster when the number n of machines varies from 4 to 20. (5) $\mathsf{PIncMiner}_{\lambda}$ can discover high quality rules validated by data quality experts. (6) Incremental rule discovery suggests new paradigms for mining rules in stages such that the users can effectively and efficiently find rules that meet their need.

8 Conclusion

The novelty of the work consists of the following. The work (1) formulates and studies the problems for incremental rule discovery in response to updates to the thresholds for support, confidence, and both; (2) provides the first algorithms for the incremental problems with boundedness relative to the batch counterpart; and (3) develops strategies for scaling with large datasets such as sampling for search lattice with accuracy guarantees and parallel incremental discovery with the parallel scalability. Our experimental study has verified that our algorithms are promising in practice.

A topic for future work is incremental discovery of top-k diversified rules that fit users' need and differ from each other. Another topic is to incrementally mine fraud-detection rules in dynamic data.

Acknowledgements

We thank Peng Liu and Yaoshu Wang for their extensive support in our experiments, and Rui Fan, Ziyan Han, and Min Xie for their feedback on an early version of this work. We are also grateful to the anonymous SIGMOD reviewers for their insightful comments. This work was supported by the ShanghaiTech Startup Fund.

References

[1] 2024. Rock. http://www.grandhoo.com/en.

[2] Ziawasch Abedjan, Patrick Schulze, and Felix Naumann. 2014. DFD: Efficient functional dependency discovery. In *CIKM*. 949–958.

[3] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases*. Addison-Wesley.

[4] Rakesh Agrawal and Ramakrishnan Srikant. 1994. Fast Algorithms for Mining Association Rules in Large Databases. In *VLDB*.

[5] Madhu V Ahluwalia, Aryya Gangopadhyay, Zhiyuan Chen, and Yelena Yesha. 2015. Target-based, privacy preserving, and incremental association rule mining. *IEEE Transactions on Services Computing* 10, 4 (2015), 633–645.

[6] Elaine Angelino, Nicholas Larus-Stone, Daniel Alabi, Margo I. Seltzer, and Cynthia Rudin. 2017. Learning Certifiably Optimal Rule Lists for Categorical Data. *J. Mach. Learn. Res.* 18 (2017), 234:1–234:78.

[7] Iyad Aqra, Norjihan Abdul Ghani, Carsten Maple, José Machado, and Nader Sohrabi Safa. 2019. Incremental algorithm for association rule mining under dynamic threshold. *Applied Sciences* 9, 24 (2019), 5398.

[8] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. 1999. Consistent Query Answers in Inconsistent Databases. In *PODS*. 68–79.

[9] Xianchun Bao, Zian Bao, Qingsong Duan, Wenfei Fan, Hui Lei, Daji Li, Wei Lin, Peng Liu, Zhicong Lv, Mingliang Ouyang, Jiale Peng, Jing Zhang, Runxiao Zhao, Shuai Tang, Shuping Zhou, Yaoshu Wang, Qiyuan Wei, and Min Xie. 2024. Rock: Cleaning Data by Embedding ML in Logic Rules. In *SIGMOD (industrial track)*. ACM.

[10] Tobias Bleifuß, Sebastian Kruse, and Felix Naumann. 2017. Efficient Denial Constraint Discovery with Hydra. *PVLDB* 11, 3 (2017), 311–323.

[11] Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. 1984. *Classification and Regression Trees*. Wadsworth.

[12] Paul Suganthan G. C., Adel Ardalan, AnHai Doan, and Aditya Akella. 2018. Smurf: Self-Service String Matching Using Random Forests. *PVLDB* 12, 3 (2018), 278–291.

[13] Loredana Caruccio and Stefano Cirillo. 2020. Incremental discovery of imprecise functional dependencies. *Journal of Data and Information Quality (JDIQ)* 12, 4 (2020), 1–25.

[14] Loredana Caruccio, Stefano Cirillo, Vincenzo Deufemia, and Giuseppe Polese. 2019. Incremental Discovery of Functional Dependencies with a Bit-vector Algorithm. In *Symposium on Advanced Database Systems (SEBD) (CEUR Workshop Proceedings, Vol. 2400)*. CEUR-WS.org.

[15] Loredana Caruccio, Stefano Cirillo, Vincenzo Deufemia, and Giuseppe Polese. 2021. Efficient discovery of functional dependencies from incremental databases. In *International Conference on Information Integration and Web Intelligence*. 400–409.

[16] Venkatesan T Chakaravarthy, Vinayaka Pandit, and Yogish Sabharwal. 2009. Analysis of sampling techniques for association rule mining. In *International Conference on Database Theory (ICDT)*. 276–283.

[17] B Chandra and Shalini Bhaskar. 2011. A new approach for generating efficient sample from market basket data. *Expert Systems with Applications* 38, 3 (2011), 1321–1325.

[18] Bin Chen, Peter Haas, and Peter Scheuermann. 2002. A new two-phase sampling based algorithm for discovering association rules. In *ACM SIGKDD international conference on Knowledge Discovery and Data Mining*. 462–468.

[19] Chyouhwa Chen, Shi-Jinn Horng, and Chin-Pin Huang. 2011. Locality sensitive hashing for sampling-based algorithms in association rule mining. *Expert Systems with Applications* 38, 10 (2011), 12388–12397.

[20] Chaofan Chen and Cynthia Rudin. 2018. An Optimization Approach to Learning Falling Rule Lists. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, Vol. 84. PMLR, 604–612.

[21] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. 2013. Discovering Denial Constraints. *PVLDB* 6, 13 (2013), 1498–1509.

[22] Kun-Ta Chuang, Ming-Syan Chen, and Wen-Chieh Yang. 2005. Progressive sampling for association rules based on sampling error estimation. In *Advances in Knowledge Discovery and Data Mining (PAKDD)*. Springer, 505–515.

[23] William W. Cohen. 1995. Fast Effective Rule Induction. In *International Conference on Machine Learning*. Morgan Kaufmann, 115–123.

[24] Ting Deng and Wenfei Fan. 2014. On the Complexity of Query Result Diversification. *ACM Trans. Database Syst.* 39, 2 (2014), 15:1–15:46.

[25] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2008. Conditional Functional Dependencies for Capturing Data Inconsistencies. *ACM Trans. Database Syst.* 33, 1 (2008), 25:1–25:49.

[26] Wenfei Fan, Floris Geerts, Jianzhong Li, and Ming Xiong. 2011. Discovering conditional functional dependencies. *TKDE* 23, 5 (2011), 683–698.

[27] Wenfei Fan, Ziyan Han, Weilong Ren Yaoshu Wang, Min Xie, and Mengyi Yan. 2024. Splitting Tuples of Mismatched Entities. *Proc. ACM Manag. Data* (2024).

[28] Wenfei Fan, Ziyan Han, Yaoshu Wang, and Min Xie. 2022. Parallel Rule Discovery from Large Datasets by Sampling. In *SIGMOD*. ACM, 384–398.

[29] Wenfei Fan, Ziyan Han, Yaoshu Wang, and Min Xie. 2023. Discovering Top-k Rules using Subjective and Objective

Criteria. *Proc. ACM Manag. Data* 1, 1 (2023), 70:1–70:29.
[30] Wenfei Fan, Muyang Liu, Shuhao Liu, and Chao Tian. 2024. Capturing More Associations by Referencing Knowledge Graphs. *PVLDB* 17, 6 (2024), 1173–1186.
[31] Wenfei Fan, Ping Lu, and Chao Tian. 2020. Unifying logic rules and machine learning for entity enhancing. *Sci. China Inf. Sci.* 63, 7 (2020).
[32] Wenfei Fan and Chao Tian. 2022. Incremental Graph Computations: Doable and Undoable. *ACM Trans. Database Syst.* 47, 2 (2022), 6:1–6:44.
[33] Wenfei Fan, Chao Tian, Yanghao Wang, and Qiang Yin. 2021. Parallel Discrepancy Detection and Incremental Detection. *PVLDB* 14, 8 (2021), 1351–1364.
[34] Wenfei Fan, Chao Tian, Ruiqi Xu, Qiang Yin, Wenyuan Yu, and Jingren Zhou. 2021. Incrementalizing Graph Algorithms. In *SIGMOD*. 459–471.
[35] Wenfei Fan, Xin Wang, Yinghui Wu, and Jingbo Xu. 2015. Association Rules with Graph Patterns. *PVLDB* 8, 12 (2015), 1502–1513.
[36] Peter A Flach and Iztok Savnik. 1999. Database dependency discovery: A machine learning approach. *AI communications* 12, 3 (1999), 139–160.
[37] Johannes Fürnkranz and Gerhard Widmer. 1994. Incremental Reduced Error Pruning. In *International Conference on Machine Learning*. Morgan Kaufmann, 70–77.
[38] Wensheng Gan, Jerry Chun-Wei Lin, Philippe Fournier-Viger, Han-Chieh Chao, and Philip S. Yu. 2019. A Survey of Parallel Sequential Pattern Mining. *ACM Trans. Knowl. Discov. Data* 13, 3 (2019), 25:1–25:34.
[39] Chang Ge, Ihab F. Ilyas, and Florian Kerschbaum. 2019. Secure Multi-Party Functional Dependency Discovery. *PVLDB* 13, 2 (2019), 184–196.
[40] Tarek F Gharib, Hamed Nassar, Mohamed Taha, and Ajith Abraham. 2010. An efficient algorithm for incremental mining of temporal association rules. *Data & Knowledge Engineering* 69, 8 (2010), 800–815.
[41] Lukasz Golab, Howard Karloff, Flip Korn, Divesh Srivastava, and Bei Yu. 2008. On generating near-optimal tableaux for conditional functional dependencies. *PVLDB* 1, 1 (2008), 376–390.
[42] Jiawei Han, Jian Pei, and Yiwen Yin. 2000. Mining Frequent Patterns without Candidate Generation. In *SIGMOD*.
[43] Xuegang Hu and Haitao Yu. 2006. The research of sampling for mining frequent itemsets. In *Rough Sets and Knowledge Technology*. Springer, 496–501.
[44] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. 1999. TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies. *The computer journal* 42, 2 (1999), 100–111.
[45] Wontae Hwang and Dongseung Kim. 2006. Improved association rule mining by modified trimming. In *IEEE International Conference on Computer and Information Technology (CIT)*. IEEE, 24–24.
[46] Caiyan Jia and Ruqian Lu. 2005. Sampling ensembles for frequent patterns. In *International Conference on Fuzzy Systems and Knowledge Discovery*. Springer, 1197–1206.
[47] Richard M Karp. 2010. *Reducibility among combinatorial problems.* Springer.
[48] Hanna Köpcke, Andreas Thor, and Erhard Rahm. 2010. Evaluation of entity resolution approaches on real-world match problems. *PVLDB* 3, 1-2 (2010), 484–493.
[49] loannis Koumarelas, Thorsten Papenbrock, and Felix Naumann. 2020. MDedup: Duplicate detection with matching dependencies. *PVLDB* 13, 5 (2020), 712–725.
[50] Sebastian Kruse and Felix Naumann. 2018. Efficient discovery of approximate dependencies. *PVLDB* 11, 7 (2018), 759–772.
[51] Clyde P. Kruskal, Larry Rudolph, and Marc Snir. 1990. A Complexity Theory of Efficient Parallel Algorithms. *Theor. Comput. Sci.* 71, 1 (1990), 95–132.
[52] Yanrong Li and Raj P Gopalan. 2004. Effective sampling for mining association rules. In *Australasian Joint Conference on Artificial Intelligence*. Springer, 391–401.
[53] Yanrong Li and Raj P Gopalan. 2005. Stratified Sampling for Association Rules Mining. In *Artificial Intelligence Applications and Innovations (AIAI)*. Springer, 79–88.
[54] Jimmy Lin, Chudi Zhong, Diane Hu, Cynthia Rudin, and Margo I. Seltzer. 2020. Generalized and Scalable Optimal Sparse Decision Trees. In *International Conference on Machine Learning (ICML)*, Vol. 119. PMLR, 6150–6160.
[55] Ester Livshits, Alireza Heidari, Ihab F. Ilyas, and Benny Kimelfeld. 2020. Approximate Denial Constraints. *PVLDB* 13, 10 (2020), 1682–1695.
[56] Stéphane Lopes, Jean-Marc Petit, and Lotfi Lakhal. 2000. Efficient discovery of functional dependencies and Armstrong relations. In *EDBT*. Springer, 350–364.
[57] Basel A Mahafzah, Amer F Al-Badarneh, and Mohammed Z Zakaria. 2009. A new sampling technique for association rule mining. *Journal of Information Science* 35, 3 (2009), 358–376.
[58] Heikki Mannila, Hannu Toivonen, and A Inkeri Verkamo. 1994. Efficient algorithms for discovering association rules. In *KDD: AAAI workshop on Knowledge Discovery in Databases*. Citeseer, 181–192.

[59] Stephen H. Muggleton and Luc De Raedt. 1994. Inductive Logic Programming: Theory and Methods. *J. Log. Program.* 19/20 (1994), 629–679.

[60] Noel Novelli and Rosine Cicchetti. 2001. Fun: An efficient algorithm for mining functional and embedded dependencies. In *ICDT*. Springer, 189–203.

[61] Thorsten Papenbrock and Felix Naumann. 2016. A Hybrid Approach to Functional Dependency Discovery. In *SIGMOD*.

[62] Srinivasan Parthasarathy. 2002. Efficient progressive sampling for association rules. In *IEEE International Conference on Data Mining*. IEEE, 354–361.

[63] Eduardo H. M. Pena, Eduardo Cunha de Almeida, and Felix Naumann. 2019. Discovery of Approximate (and Exact) Denial Constraints. *PVLDB* 13, 3 (2019), 266–278.

[64] Chaoqin Qian, Menglu Li, Zijing Tan, Ai Ran, and Shuai Ma. 2023. Incremental discovery of denial constraints. *VLDB J.* 32, 6 (2023), 1289–1313.

[65] J. Ross Quinlan. 1986. Induction of Decision Trees. *Mach. Learn.* 1, 1 (1986), 81–106.

[66] J. Ross Quinlan. 1993. *C4.5: Programs for Machine Learning.* Morgan Kaufmann.

[67] J. Ross Quinlan. 2004. Data Mining Tools See5 and C5.0.

[68] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *PVLDB* 10, 11 (2017), 1190–1201.

[69] Matteo Riondato and Eli Upfal. 2015. Mining frequent itemsets through progressive sampling with rademacher averages. In *SIGKDD*. ACM, 1005–1014.

[70] Cynthia Rudin. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nat. Mach. Intell.* 1, 5 (2019), 206–215.

[71] Cynthia Rudin, Benjamin Letham, and David Madigan. 2013. Learning theory analysis for association rules and sequential event prediction. *J. Mach. Learn. Res.* 14, 1 (2013), 3441–3492.

[72] Nandlal L Sarda and NV Srinivas. 1998. An adaptive algorithm for incremental mining of association rules. In *International Workshop on Database and Expert Systems Applications (Cat. No. 98EX130)*. IEEE, 240–245.

[73] Philipp Schirmer, Thorsten Papenbrock, Ioannis K. Koumarelas, and Felix Naumann. 2020. Efficient Discovery of Matching Dependencies. *ACM Trans. Database Syst.* 45, 3 (2020), 13:1–13:33.

[74] Philipp Schirmer, Thorsten Papenbrock, Sebastian Kruse, Felix Naumann, Dennis Hempfing, Torben Mayer, and Daniel Neuschäfer-Rube. 2019. DynFD: Functional Dependency Discovery in Dynamic Datasets. In *EDBT*.

[75] Rohit Singh, Venkata Vamsikrishna Meduri, Ahmed K. Elmagarmid, Samuel Madden, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Armando Solar-Lezama, and Nan Tang. 2017. Synthesizing Entity Matching Rules by Examples. *PVLDB* 11, 2 (2017), 189–202.

[76] Shaoxu Song and Lei Chen. 2009. Discovering matching dependencies. In *CIKM*.

[77] Apache Spark. 2024. Frequent Pattern Mining. https://spark.apache.org/docs/latest/ml-frequent-pattern-mining.html.

[78] Ramakrishnan Srikant and Rakesh Agrawal. 1996. Mining Sequential Patterns: Generalizations and Performance Improvements. In *EDBT*. Springer, 3–17.

[79] Zijing Tan, Ai Ran, Shuai Ma, and Sheng Qin. 2020. Fast Incremental Discovery of Pointwise Order Dependencies. *PVLDB* 13, 10 (2020), 1669–1681.

[80] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. ArnetMiner: Extraction and Mining of Academic Social Networks. In *KDD*. 990–998.

[81] Wannasiri Thurachon and Worapoj Kreesuradej. 2021. Incremental association rule mining with a fast incremental updating frequent pattern growth algorithm. *IEEE Access* 9 (2021), 55726–55741.

[82] Hannu Toivonen. 1996. Sampling large databases for association rules. In *VLDB*. 134–145.

[83] Pauray SM Tsai, Chih-Chong Lee, and Arbee LP Chen. 1999. An efficient approach for incremental association rule mining. In *PAKDD*. Springer, 74–83.

[84] Jeffrey D. Ullman. 2000. A Survey of Association-Rule Mining. In *International Conference on Discovery Science*. Springer, 1–14.

[85] Leslie G. Valiant. 1990. A Bridging Model for Parallel Computation. *Commun. ACM* 33, 8 (1990), 103–111.

[86] Flavian Vasile, Adrian Silvescu, Dae-Ki Kang, and Vasant G. Honavar. 2006. TRIPPER: Rule Learning Using Taxonomies. In *PAKDD*.

[87] Tong Wang, Cynthia Rudin, Finale Doshi-Velez, Yimin Liu, Erica Klampfl, and Perry MacNeille. 2017. A Bayesian Framework for Learning Rule Sets for Interpretable Classification. *J. Mach. Learn. Res.* 18 (2017), 70:1–70:37.

[88] G. I. Webb and S. Zhang. 2005. k-Optimal Rule Discovery. *Data Mining and Knowledge Discovery* 10, 1 (2005), 39–79.

[89] Catharine M. Wyss, Chris Giannella, and Edward L. Robertson. 2001. FastFDs: A Heuristic-Driven, Depth-First Algorithm for Mining Functional Dependencies from Relation Instances - Extended Abstract. In *DaWak*.

[90] Ying Yan, Liang Jeff Chen, and Zheng Zhang. 2014. Error-bounded sampling for analytics on big sparse data. *PVLDB* 7, 13 (2014), 1508–1519.

[91] Hong Yao, Howard J. Hamilton, and Cory J. Butz. 2002. FD_Mine: Discovering Functional Dependencies in a Database

Using Equivalences. In *International Conference on Data Mining (ICDM)*. IEEE Computer Society, 729–732.

[92] Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. 1997. New Algorithms for Fast Discovery of Association Rules. In *KDD*. AAAI Press, 283–286.

[93] Chengqi Zhang, Shichao Zhang, and Geoffrey I Webb. 2003. Identifying approximate itemsets of interest in large databases. *Applied Intelligence* 18 (2003), 91–104.

[94] Yunjia Zhang, Zhihan Guo, and Theodoros Rekatsinas. 2020. A Statistical Perspective on Discovering Functional Dependencies in Noisy Data. In *SIGMOD*. 861–876.

[95] Yanchang Zhao, Chengqi Zhang, and Shichao Zhang. 2006. Efficient Frequent Itemsets Mining by Sampling. *AMT* 138 (2006), 112–7.

A Proof of Proposition 1

Proposition 1: *The decision version of the optimization problem of sampling the search lattice, as formulated above, is* NP*-complete.* □

It suffices to show that the decision version of the sampling problem is in NP, and that a special case of the sampling problem is already NP-hard. The special case treats the covering radius K_p for each sample p as 1. If these hold, the sampling problem is NP-complete.

Sampling problem. Given a lattice $\mathcal{L}$, a storage budget B, a coverage radius K_p for each node $p \in \mathcal{L}$, the sampling problem is to decide whether there exists a set of sampled nodes $\mathcal{P} \subseteq \mathcal{L}$ such that

- every node in $\mathcal{L}$ is covered by at least one sampled node, *i.e.,*
$$\forall v \in \mathcal{L},\ \exists p \in \mathcal{P},\ v \in N(p, K_p),$$
where $N(p, K_p)$ represents the set of nodes covered by a sampled node p with radius K_p; and
- the number of sampled nodes is at most B, *i.e.,* $|\mathcal{P}| \leq B$.

The set $N(p, K_p)$ is defined as follows. A node v is in $N(p, K_p)$ if and only if v is an immediate successor of any of the first K_p immediate predecessors of p in the lattice $\mathcal{L}$; let $q \to v$ denote that v is the immediate successor of q; then
$$N(p, K_p) = \{v \in \mathcal{L} \mid \exists\, q \in \text{Pre}(p)[1 : K_p],\ q \to v\}.$$
Here $\text{Pre}(p)[1 : K_p]$ denotes the first K_p predecessors of p.

Upper bound. An NP algorithm for the sampling problem works as follows: Guess a set $\mathcal{P} \subseteq \mathcal{L}$ of sampled nodes with radius K_p for each $p \in \mathcal{P}$, and verify that $\mathcal{P}$ satisfies the coverage and budget constraints. It suffices to show that the verification step can be done in PTIME. For if it holds, then the sampling problem is in NP.

We show that verification is in PTIME. It works as follows.

- Verify the storage budget constraint: If $|\mathcal{P}| > B$, the algorithm returns `false`, as the storage budget constraint is violated.
- Verify the coverage constraint: For each node $v \in \mathcal{L}$, check whether v is covered by at least one node $p \in \mathcal{P}$. If any node v is not covered, the algorithm returns `false`.
- If both constraints are satisfied, the algorithm returns `true`.

The algorithm runs in PTIME: (a) Counting the size of $\mathcal{P}$ takes $O(|\mathcal{P}|)$ time. (b) For each node $v \in \mathcal{L}$, the algorithm iterates over all sampled nodes $p \in \mathcal{P}$ to check whether $v \in N(p, K_p)$. Each sample p covers at most $|\mathcal{L}|$ nodes. This step takes at most $O(|\mathcal{L}|^2 \cdot |\mathcal{P}|)$ time. (c) The overall complexity is $O(|\mathcal{L}|^2 \cdot |\mathcal{P}|)$. Thus the algorithm is in PTIME, and the sampling problem is in NP.

Lower bound. We show that a simplified sampling problem is NP-hard by reduction from the Set Cover problem, which is NP-complete [47]. The Set Cover problem is to decide, given a universe $U = \{u_1, u_2, \ldots, u_n\}$ of n elements, a collection of subsets $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ such that $\bigcup_{i=1}^{m} S_i = U$,

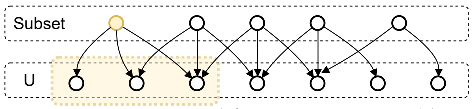


Fig. 12. Lattice constructed from Set Cover problem input

and a budget k, whether there exists a sub-collection $\mathcal{S}' \subseteq \mathcal{S}$ with $|\mathcal{S}'| \leq k$ that covers U.

We consider a special case of the sampling problem by treating the radius K_p for each sample node p as 1, referred to as Sampling in the sequel. In this special case, the cover relation can be interpreted as follows: node v_j covers node v_i if and only if v_j is an immediate predecessor of v_i in the lattice. As illustrated in Figure 12, node v_j covers nodes in the highlighted area in the "U" layer.

We reduce an instance of Set Cover to an instance of the special case of the sampling problem as follows.

- We define a lattice structure as shown in Figure 12: (1) the bottom "U" layer represents the universe U in Set Cover, where each node uniquely corresponds to one element in U. (2) The middle "Subset" layer represents the subsets $\mathcal{S}$ in Set Cover, such that each node uniquely corresponds to one subset in $\mathcal{S}$. Edges are added between nodes in the two layers to represent membership relation in Set Cover. (3) A top-level "Root" node is added as a common predecessor of all nodes in the "Subset" layer.
- We set the storage budget in the sampling problem $B = k + 1$, where k is the budget constraint in the Set Cover problem; we add 1 to k to cover the "Root" node remarked above.

From the construction above we can deduce the following.

- A one-to-one mapping $L_U : U \longrightarrow V_U$, such that each element $u_i \in U$ uniquely maps to a node v_i in the "U" layer.
- A one-to-one mapping $L_S : \mathcal{S} \longrightarrow V_S$, such that each subset $S_j \in \mathcal{S}$ uniquely maps to a node v_j in the "Subset" layer.
- There are edges between nodes in the "Subset" and "U" layer, such that $L_S(S_j) \longrightarrow L_U(u_i)$ iff $u_i \in S_j$ in the Set Cover instance.
- The root is a common predecessors of all "Subset" layer nodes in $\mathcal{L}_S$, with corresponding edges. Thus lattice $\mathcal{L}$ has three layers with nodes in V_U and V_S, and the root at the top level.

The reduction is obviously in PTIME.

Below we show that there exists a cover $\mathcal{S}' \subseteq \mathcal{S}$ with $|\mathcal{S}'| \leq k$ that covers U for Set Cover if and only if there exists a set of sampled nodes $\mathcal{P} \subseteq \mathcal{L}$ that satisfies the coverage and budget constraints.

$\Rightarrow$ First, assume the existence of a cover $\mathcal{S}'$ for Set Cover. We give a set of sampled nodes $\mathcal{P}$ for Sampling as follows. (1) For each subset $S_j \in \mathcal{S}'$, find its corresponding node in the lattice $v_j = L_S^{-1}(S_j)$, and add v_j in $\mathcal{P}$. (2) Add the root node into $\mathcal{P}$.

One can verify that $\mathcal{P}$ satisfies both constraints in the sampling problem as follows: (a) for each node v_j in "U" layer, its corresponding element in U is covered by a subset in $S_j \in \mathcal{S}'$. By the construction above, each S_j corresponds to a sampled node p_j in $\mathcal{P}$, which covers v_j. Thus, all nodes in layer "U" are covered by $\mathcal{P}$. (b) Nodes in the "Subset" layer are covered by the root node. (c) $|\mathcal{P}| = |\mathcal{S}'| + 1 <= k + 1 = B$, satisfying the budget constraint.

$\Leftarrow$ Conversely, assume the existence of a set $\mathcal{P}$ of sampled nodes for Sampling. We give a cover $\mathcal{S}'$ for Set Cover: for each sampled node $v_j \in \mathcal{P}$, if v_j is in the "Subset" layer of $\mathcal{L}$, then find its corresponding subset $S'_j = L_S^{-1}(v_j)$ and put S'_j into $\mathcal{S}'$. Otherwise, if v_j is in the other two layers, skip and examine the next node.

One can verify that $\mathcal{S}'$ satisfies both constraints in Set Cover. More specifically, (a) by the coverage constraint in Sampling, every node v_i in the "U" layer is covered by at least one sampled node p_j in $\mathcal{P}$. Moreover, by the definition of cover, v_i must be covered by a node in "Subset" layer. Since every node in "U" layer corresponds to an element in U, and every sampled node in "Subset" layer corresponds to a subset $\mathcal{S}' \in \mathcal{S}$ in Set Cover, every node $u \in U$ is covered by a subset in $\mathcal{S}$. (b) By the construction, $|\mathcal{S}'| \leq |\mathcal{P}| - 1 \leq B - 1 = k$ (note that $\mathcal{P}$ must contain the root node to cover the full lattice), satisfying the Set Cover budget constraint. □

Received October 2024; revised January 2025; accepted February 2025