Discovering Top-k Interesting Rules

Wenfei Fan[1,2,3] Ziyan Han[3] Yaoshu Wang[1]
Shenzhen Institute of Computing Sciences[1] University of Edinburgh[2] Beihang University[3]
wenfei@inf.ed.ac.uk,hanzy@act.buaa.edu.cn,yaoshuw@sics.ac.cn

ABSTRACT

This paper studies two questions about rule discovery. What rules are interesting? How can we discover top-ranked interesting rules? We consider *entity enhancing rules* (REEs), which can embed machine learning classifiers as predicates, and subsume conditional functional dependencies, denial constraints and matching dependencies as special cases. We propose an interestingness model in terms of both objective measures such as support and confidence, and subject measures for user preference and interests; we show how to learn the subjective model and weight vectors via active learning. Based on the model, we develop a top-k algorithm to discover interesting REEs, and an any-time algorithm for successive interesting-rule discovery via lazy evaluation. We parallelize these algorithms such that they guarantee to reduce runtime when more processors are used. Using real-life and synthetic datasets, we show that the algorithms are able to find interesting rules and speed up conventional rule-discovery methods by 8.4? times on average.

1 INTRODUCTION

Dirty data has been a longstanding challenge for decades. Real-life data is often dirty, *e.g.,* attributes of the same entity may carry inconsistent or conflict values, different entities may be mismatched into one, and the same entity bears different representations and is treated as distinct ones. Dirty data is costly, *e.g.,* Gartner reports that poor data quality is responsible for an average of $15 million per year in losses for organizations [28], and IBM estimates that dirty data costs the US $3.1 trillion in 2016 alone (cf. [59]).

To improve data quality, a variety of logic rules have been studied for *entity resolution (ER)* and *conflict resolution (CR)*, *e.g.,* matching dependencies (MDs) [5, 7, 17] for ER, and denial constraints (DCs) [4] and conditional functional dependencies (CFDs) [18, 20] for CR. Here ER is to identify tuples that refer to the same real-world entity, and CR is to resolve inconsistencies pertaining to the entities. There has also been recent effort to embed machine learning (ML) classifiers as predicates in logic rules [22], to unify ML-based and rule-based methods. When provided with high-quality rules, one can detect value conflicts and entity mismatches, fix the errors, and interpret the reasons behind the errors and their corrections.

To make practical use of the rule-based methods, it is a must to be able to discover interesting rules from real-life datasets. In the big data era, it is unrealistic to solely rely on domain experts to write or find rules manually. Several algorithms have been developed for discovering functional dependencies (FDs) [6, 35, 38, 54, 74, 78], DCs [13, 47, 55], MDs [65, 69] and CFDs [11, 19]. However, a major problem encountered in practice is For example, on a small dataset with only 27 attributes and 368 tuples, there are 128,726 FDs found [53]. The practitioners are often overwhelmed by the large number of rules and have to spend a significant amount of time to manually inspect and select rules that are interesting to them. This staggering cost hampers the applicability of the rule-based methods.

Discovering interesting rules. We propose the following.

(1) Interestingness. Conventional interestingness measures include, *e.g.,* support and confidence, for how often the rules can be applied and how strong the associations between their preconditions and consequences are. In practice, however, these "universal" objective measures often do not suffice. In our experience, practitioners in industry often have accumulated "business rules" from their practice; they already know quite a few rules that have high support and confidence. What they want from rule discovery are rules that are "surprising" to them, *i.e.,* sensible rules that they do not yet know, to complement those they have already got. This suggests that we consider "subjective measures" in the interestingness model, to capture the preference and interests of different users.

In light of this, we propose a model to characterize the interestingness of rules, in terms of (a) conventional objective measures, and (b) new subjective measures for users preference and interests. The two parts bear various weights to serve different users. Experienced users may assign a higher weight to the subjective measure in favor of surprising rules, while novice users may opt to prioritize objective measure to find rules with high support and confidence.

(2) Top-k algorithm. Based on the interestingness model, we develop an efficient algorithm for discovering top-ranked rules. A brute-force approach is to first find all the rules that hold on a dataset, and then sort the rules and return top-k ones for a predefined constant k. Can we do better? Observe that when we use a search engine, we do not want it to first enumerate all matches of our keywords and then return the top-ranked ones. Analogous to search, users typically want to efficiently find top-k rules.

We formulate the top-k rule discovery problem, and develop a top-k algorithm as a solution. The algorithm terminates early as soon as it finds top-k rules, without incurring the cost of inspecting all rules that hold on the dataset. To the best of our knowledge, no such top-k discovery algorithm is yet available for data quality rules such as functional dependencies (FDs), DCs, CFDs and MDs.

(3) Anytime algorithm. If the top-k rules do not meet users' need, they often want to continue to find the next top-k ones. We provide an anytime-algorithm to find the next top-k rules if needed, via lazy evaluation. By repeatedly running the algorithm, users can find all the rules that hold on a dataset in the order of interestingness.

(4) Parallelization. To scale with large datasets, we parallelize the top-k algorithm and the anytime algorithm across a cluster of machines. We show that the algorithms are parallel scalable [40], *i.e.,* the algorithms guarantee to reduce the parallel runtime when more machines are used. Hence in theory, the algorithms are able to efficiently discover rules even when the datasets grow big, as long as we add more computing resources when needed.

Contributions & organizations. This paper explores a new approach to discovering interesting rules, for experienced users and novice users. As a testbed of our proposed methods, we consider entity enhancing rules (REEs) [22], since REEs (a) embed ML classifiers in logic rules, (b) unify ER and CR with the same rules, (c) are collectively defined across multiple relations, and (d) subsume DCs, CFDs and MDs as special cases. We will review REEs in Section 2.

(1) Interestingness model (Section 3). We propose (a) an interestingness model for REEs, and (b) an active learning method to determine the subjective model and weight vectors. In addition, we show that the conventional notion of support does not fit collective rules such as REEs, *i.e.,* it is no longer anti-monotonic; in light of this, we revise the notion for REEs and show its anti-monotonicity.

(2) Discovering interesting rules. We develop a set of techniques:

- a top-k algorithm for discovering REEs (Section 4),
- an anytime algorithm via lazy evaluation (Section 5), and
- their parallelization with parallel scalability (Section 6).

In particular, we introduce techniques to (a) learn an interestingness upper bound for early termination, and (b) handle collective REEs with multiple tuple variables and their embedded ML predicates.

(3) Experimental study (Section 7). Using real-life and synthetic datasets, we empirically find the following. On average, (a) top-k REEs discovery speeds up conventional methods by 8.4 times. It takes 667s on DBLP with 18 attributes and 1,799,559 tuples for k = 10 with 20 machines, versus 8726s by traditional methods. (b) The lazy evaluation strategy makes the anytime algorithm 4.85 times faster than the top-k one when users want the 4th top-10 REEs. (c) Our interestingness model is 25.5% more accurate than the state-of-the-art language models, and its subjective measures improves the accuracy from 0.48 to 0.76. (d) The algorithms are parallel scalable; they are 3.6 times faster using 20 machines instead of 4.

To the best of our knowledge, this work makes the first effort to train and employ subjective models for discovering interesting data quality rules. It also provides the first top-k and anytime algorithms for such rules, with the parallel scalability. Since REEs subsume DCs, FDs, CFDs and MDs, our algorithms can also be those rules.

Related work. The related work is categorized as follows.

Interestingness models. The need for considering both objective measures and subjective measures has long been recognized in the data mining community, to reflect users' universal and individualistic preference, respectively (see [30] for a survey). However, we are not aware of any discovery algorithms for FDs, DCs, CFDs and MDs that take subjective measures into account. For instance, while MDedup [37] employs several objective features for MDs in feature engineering and adopts the ML regression model Gaussian Process to learn a score for each MD, it aims to discover MDs with high F-measures, not for users' surprisingness or unexpectedness. While ML models [15, 67] were designed to learn the representations of rules, they do not consider how to learn rule interestingness.

This work differs from the prior work as follows. (1) We propose an interestingness model designated for data quality rules that combines both objective and subjective measures, the first of the kind for REEs, DCs, CFDs and MDs. (2) We propose to use active learning and pairwise ranking strategies to learn the interestingness of rules, to catch the users' background knowledge.

Rule discovery. A number of discovery algorithms have been developed for data quality rules, classified as follows. (1) *Levelwise search.* TANE [35], FUN [52], FD_mine [76] discover FDs based on a lattice structure. Depmine [48] develops agree sets to accelerate the search, HyFD [54] combines sampling techniques and lattice traversal to prune non-FDs, and DynFD [66] mines FDs in dynamic datasets. The levelwise methods have also been extended to CFDs discovery, including CTANE [19] and tableau generation [32]. For ER rules, [69] mines thresholds of similarity functions for discovering MDs. SMFD [29] proposes a top-down based framework to discover and validate FDs in a secured multi-party scenario. (2) *Depth-first search.* DFD [2] adopts the depth-first search (random walk) in the lattice structure. FastFDs [74] improves Depmine by using different sets to mine FDs. For CFDs and DCs, depth-first search approaches also apply, *e.g.,* FastCFDs [19] extends FastFDs to mine CFDs, while FastDC [13], Hydra [9], DCFinder [55] and ADCMiner [47] find valid DCs based on evidence sets and min-cover algorithms. Although DCs support multiple relation atoms [13] by definition, all its discovery algorithms [9, 47, 55] restrict their settings to bi-variable DCs discovery only. (3) *Hybrid approaches.* HyMD [65] makes use of both levelwise and depth-first search, and discovers MDs with pre-computed index structures. MDedup [37] approaches ER by selecting useful MDs discovered by HyMD. (4) *Learning-based approaches.* [24] utilizes inductive learning to find FDs. AutoFD [78] adopts structure learning (*i.e.,* graphical lasso) to discover FDs. [68] and [36] adopt the rule learning strategies to find MDs with similarity functions (see [12] for more ER solvers). (5) *Top-k discovery.* Closer to this work are top-k algorithms for discovering association rules in relations [72] and graphs [23].

This work differs from the prior work in the following. (1) We study top-k rule discovery based on a new interestingness model, not to find all rules on a dataset subject to *e.g.,* support. It differs from [23, 72] in the use of different ranking criteria (interestingness) and thus different early-termination strategies. (2) We extend our top-k algorithm to an anytime algorithm, in order for users to find the next top-k results when needed, without re-discovering starting from scratch. No existing work has considered anytime rule discovery. (3) It is more challenging to discover REEs than DCs, CFDs and MDs, since REEs can embed ML predicates, and are collectively defined across multiple relations. For instance, the traditional notion of support is no longer anti-monotonic for REEs.

ML models. Various ML models have been studied for data cleaning. (1) *Models for ER*, via SVM and logistic regression (see [31] for a survey), unsupervised learning, *e.g.,* ZeroER [73], and deep learning, *e.g.,* Ditto [45], DeepMatcher [51], DeepER [16], AutoEM [79], GraphER [42], MPM [26]. (2) *Models for CR*, including HoloClean [61], HoloDetect [34], Raha [49] and SLiMFast [62], for error detection, error correction and data fusion. (3) *Models for similarity checking*, by employing popular language models such as BERT-based models [14, 41, 46, 60], XLNet [75] and GPT [10, 58].

This work is not to develop another ML model. Instead, we show how to leverage existing well-trained ML models by embedding them as ML predicates in REEs. As will be seen in Example 1, one can even plug in a link prediction ML models into REEs to reveal the "hidden" associations between entities, to facilitate ER and CR.

Parallel rule discovery. Parallel algorithms have been developed for rule discovery. [27, 43] employ massive parallelism to discover FDs,

tid	pid	address	zipcode	#beds	#baths	selling_price($)
t_1	p_1	67 old head RD, Sydney	2029	2	2	0.85 M
t_2	p_2	NO.67 old head road, SYD	2029	2	2	0.85 M
t_3	p_3	15 Pitt street, Pitt point	2022	5	3	1.50 M
t_4	p_4	15 Pitt street	2022	5	3	1.20 M

Table 1: Example Property **relation** D_1

tid	cid	pid	renter	tenant	monthly_rent	rent_start_year	rent_end_year
t_5	c_1	p_1	Michael	Leo	560	2016	2020
t_6	c_2	p_2	Mich.	Leo Z.	560	2016	2020
t_7	c_3	p_3	Ada	Mike	1,300	2014	2016
t_8	c_4	p_3	Ada	Connie	1,500	2016	2018
t_9	c_5	p_4	Ada	Bob	1,500	2018	2020

Table 2: Example RentalContract **relation** D_2

but they do not consider communication cost. [44] discovers FDs in a distributed setting but only returns local FDs. [63] extends FastFD by minimizing communication cost. [64] proposes a distributed framework for discovering both FDs and DCs. SMFD [29] performs FD discovery based on the lattice structure in a distributed manner, but it only focuses on how to enforce privacy constraints.

This work is among the first discovery algorithms for relational data quality rules that guarantee the parallel scalability, *i.e.,* the execution time is guaranteed to be reduced if more machines are used, when both computational and communication costs are considered.

2 ENTITY ENHANCING RULES

We next review entity enhancing rules (REEs) defined in [22].

We define REEs over a database schema $\mathcal{R} = (R_1, \ldots, R_m)$. Each R_j is a schema of the form $R(A_1 : \tau_1, \ldots, A_n : \tau_n)$, where A_i is an attribute of type τ_i. A relation D of R is a set of tuples having attributes A_i of R ($i \in [1, n]$) with values from the domains of τ_i. We assume *w.l.o.g.* that each tuple t in D has an id attribute, which uniquely identifies the entity that t represents. An instance $\mathcal{D}$ of $\mathcal{R}$ is a collection $(D_1, \ldots, D_m)$, where D_i is a relation of R_i ($i \in [1, m]$).

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

$$p ::= R(t) \mid t.A \oplus c \mid t.A \oplus s.B \mid \mathcal{M}(t[\bar{A}], s[\bar{B}]),$$

where $\oplus$ is an operator in $\{=, \neq\}$. Following tuple relational calculus [3], (a) $R(t)$ is a *relation atom* over $\mathcal{R}$, where $R \in \mathcal{R}$, and t is a *tuple variable bounded by* $R(t)$; (b) $t.A$ denotes an attribute of t when t is bounded by $R(t)$ and A is an attribute in R; (c) $t.A \oplus c$ is *a constant predicate* when c is a value in the domain of A; and (d) $t.A \oplus s.B$ compares *compatible* attributes $t.A$ and $s.B$, *i.e.,* tuple t (resp. s) is bounded by $R(t)$ (resp. $R'(s)$), and $A \in R$ and $B \in R'$ have the same type. Moreover, (e) $\mathcal{M}(t[\bar{A}], s[\bar{B}])$ is *an ML predicate* where $t[\bar{A}]$ and $s[\bar{B}]$ are vectors of pairwise compatible attributes.

Here $\mathcal{M}$ can be any existing ML model that returns a Boolean value, *e.g.,* $\mathcal{M}_{\text{reg}} \geq \delta$ for a regression model $\mathcal{M}_{\text{reg}}$ and a bound δ. In this paper, we consider $\mathcal{M}$ such as (1) NLP models, *e.g.,* Bert [14], for text classification; (2) ER models and link prediction models, *e.g.,* ditto [45] and DeepMatcher [51], to reveal "hidden" associations between tuples across relations, which are not connected by, *e.g.,* keys and foreign keys; and (3) models for data fusion, error detection and correction, *e.g.,* HoloClean [61] and HoloDetect [34].

REEs. An *entity enhancing rule* (REE) φ over $\mathcal{R}$ is defined as

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

where X is a conjunction of *predicates* over $\mathcal{R}$, and p_0 is a predicate over $\mathcal{R}$ such that all tuple variables in φ are bounded in X. We refer to X as the *precondition* of φ and p_0 as the *consequence* of φ.

Example 1: Consider the following example from a (simplified) realty database with self-explained schemas Property (pid, address, zipcode, #beds, #baths, selling_price) and RentalContract (cid, pid, renter, tenant, monthly_rent, rent_start_year, rent_end_year).

Below are example REEs over the database schema.

(1) φ_1 : Property(t_a) $\wedge$ Property(t_b) $\wedge$ $\mathcal{M}_{\text{addr}}(t_a.\text{address}, t_b.\text{address})$ $\wedge$ RentalContract(s_a) $\wedge$ RentalContract(s_b) $\wedge$ $t_a.\text{pid} = s_a.\text{pid} \wedge t_b.\text{pid} = s_b.\text{pid} \wedge s_a.\text{rent\_start\_year} = s_b.\text{rent\_start\_year} \wedge s_a.\text{rent\_end\_year} = s_b.\text{rent\_end\_year} \wedge \mathcal{M}_{\text{name}}(s_a[\bar{A}], s_b[\bar{A}]) \rightarrow t_a.\text{pid} = t_b.\text{pid}$. Here $\mathcal{M}_{\text{addr}}$ is an ML model for checking the closeness of addresses, $\mathcal{M}_{\text{name}}$ checks the semantic similarity on names, and $\bar{A}$ denotes (renter, tenant) in relation RentalContract.

Intuitively, φ_1 says that two properties can be identified if (a) their addresses match, (b) renters and tenants are "similar", and (c) rental periods are the same. It uses ML models $\mathcal{M}_{\text{addr}}$ and $\mathcal{M}_{\text{name}}$ to check semantic closeness of text attributes, *e.g.,* "Michael" vs. "Mike".

(2) φ_2: Property(t_a) $\wedge$ RentalContract(s_a) $\wedge$ RentalContract(s_b) $\wedge$ $\mathcal{M}_{\text{rent}}(t_a, s_a) \wedge s_a.\text{rent\_start\_year} \neq s_b.\text{rent\_start\_year} \wedge t_a.\text{pid} = s_b.\text{pid} \rightarrow s_a.\text{monthly\_rent} \neq s_b.\text{monthly\_rent}$. Here we use a link prediction model $\mathcal{M}_{\text{rent}}(t_a, s_a)$ to predict whether the tenant in s_a may rent property t_a (note that RentalContract might not record the fact "s_a rents t_a" yet). Intuitively, it says that rent fluctuates at different times, since t_a is the same property as in s_b.

(3) φ_3 : Property(t_a) $\wedge$ Property(t_b) $\wedge$ $X \rightarrow \mathcal{M}_{\text{addr}}(t_a.\text{address}, t_b.\text{address})$, where $\mathcal{M}_{\text{addr}}$ is the address ML model used in φ_1, $X = \bigwedge_{A_s \in \mathcal{T}} t_a.A_s = t_b.A_s$ and $\mathcal{T}$ denotes a designated set of attributes in Property (not shown in the simplified schema), including built-up area, floor, and neighborhood information. Here conditions in X interpret the prediction of $\mathcal{M}_{\text{addr}}(t_a.\text{address}, t_b.\text{address})$ in logic. □

As shown in Example [22], REEs subsume as special cases MDs for ER (by using ML models to check similarity), and FDs, CFDs and DCs for CR. They may carry multiple relation variables for, *e.g.,* collective ER [8], beyond bi-variable rules. Moreover, REEs may embed state-of-the-art ML models, beyond DCs, CFDs and MDs.

Semantics. Consider an instance $\mathcal{D}$ of $\mathcal{R}$. A *valuation* h of tuple variables of φ in $\mathcal{D}$, or simply a valuation of φ, is a mapping that instantiates t in each $R(t)$ with a tuple in a relation D of $\mathcal{D}$.

We say that h *satisfies* a predicate p, written as $h \models p$, if the following are satisfied: (1) If p is a relation atom $R(t)$, $t \oplus c$ or $t.A \oplus s.B$, then $h \models p$ is interpreted as in tuple relational calculus following the standard semantics of first-order logic [3]. (2) If p is $\mathcal{M}(t[\bar{A}], s[\bar{B}])$, then $h \models p$ if $\mathcal{M}$ predicts true on $(h(t)[\bar{A}], h(s)[\bar{B}])$.

Given a conjunction X of predicates, we say $h \models X$ if for *all* predicates p in X, $h \models p$. Given an REE φ, we write $h \models \varphi$ such that if $h \models X$, then $h \models p_0$. 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 $\mathcal{D}$, $h \models \varphi$. That is, REEs have universal semantics. We say that $\mathcal{D}$ *satisfies* a set Σ of REEs, denoted by $\mathcal{D} \models \Sigma$, if for all $\varphi \in \Sigma$, $\mathcal{D} \models \varphi$.

Example 2: Continuing with Example 1, assume that $\mathcal{D}$ consists of two relations D_1 and D_2 of schemas Property and RentalContract, shown in Tables 1 and 2, respectively. Consider valuation h_2: $t_3 \mapsto t_a$, $t_8 \mapsto s_a$ and $t_7 \mapsto s_b$. It satisfies REE φ_2, since Mike rented property p_3 earlier than Connie and thus, he pays a cheaper rent.

As another example, consider the valuation h_1 of REE φ_1 that has mappings: $t_1 \mapsto t_a$, $t_2 \mapsto t_b$, $t_5 \mapsto s_a$ and $t_6 \mapsto s_b$. It helps us identify that t_1 and t_2 are indeed the same property. □

3 LEARNING INTERESTINGNESS

In this section we study the interestingness of REEs. We first present objective measures and subjective measures (Section 3.1). We then propose our interesting model (Section 3.2). Finally, we propose an active-learning method to learn the subjective model and the weights of the measures in the interestingness model (Section 3.3).

3.1 Interestingness Measures

To find truly useful REEs for users, we consider (a) objective measures, which are based only on the datasets and are "universal" to different users; and (b) subjective measures, which are based on both the data and the users, including the users' background, preference and interests, and may *vary* for different groups of users.

3.1.1 Objective Measures. We start with objective measures.

Support. Support measures how frequently an REE can be applied. For collective rules that are defined across multiple relations such as REEs, the conventional notion of support has to be revised.

To see this, we first define an order on REEs. Given two REEs $\varphi: X \to p_0$ and $\varphi': X' \to p_0$ that have the same consequence p_0, we say that φ has a *lower order* than φ', denoted by $\varphi \preceq \varphi'$, if $X \subseteq X'$. Intuitively, φ is less restrictive than φ'.

One might think that given a dataset $\mathcal{D}$ of database schema $\mathcal{R}$, support for REEs $\varphi: X \to p_0$ over $\mathcal{R}$ is simply the number of distinct valuations h of φ in $\mathcal{D}$ such that $h \models X$. This is the conventional definition of support commonly used in data quality rules on a *single* relation, *e.g.,* FDs, CFDs, etc. It satisfies the *anti-monotonicity* on single relations, *i.e.,* if $\varphi \preceq \varphi'$, then the support of φ is at least that of φ'. Unfortunately, for *collective* rules involving *multiple relations*, this definition does not work. To see why it is the case, consider the following from the realty database in Example 1.

Example 3: Let X be Property$(t_a) \wedge t_a$.zipcode = 2022 and p_0 be t_a.selling_price = 1.50M. Consider two REEs φ and φ', $\varphi: X \to p_0$ and $\varphi': X' \to p_0$, where $X' = X \wedge$ RentalContract$(s_a) \wedge t_a$.pid = s_a.pid. Clearly, $\varphi \preceq \varphi'$ since $X \subset X'$. However, if the conventional support definition is applied, then the support of φ is 1 by $t_3 \mapsto t_a$, while the support of φ' is 2, since both $(t_3, t_7) \mapsto (t_a, s_a)$ and $(t_3, t_8) \mapsto (t_a, s_a)$, violating the anti-monotonicity. Intuitively, this is because in collective rules, a tuple in a relation can join with multiple tuples in another relation, yielding a larger "support". □

To fix this, we revise the notion of support and establish its anti-monotonicity, a property that is critical to reducing the search space when discovering rules. To the best of our knowledge, this is among the first anti-monotonicity for collective data quality rules.

For the ease of illustration, we assume predicates in this section involve two tuple variables, *i.e.,* $t.A \oplus s.B$ or $\mathcal{M}(t[\bar{A}], s[\bar{B}])$, while all notations extend naturally to predicates with one tuple variable.

We use the following notions. Given a predicate p, we define an REE φ_p to verify whether two tuples satisfy p: $R(t) \wedge R'(s) \to p$, where t and s (of relation schema R and R', respectively) are the tuple variables used in p. Let H_p be the set of valuations of φ_p in $\mathcal{D}$. We define the *support set* of p on $\mathcal{D}$, denoted by spset$(p, \mathcal{D})$, as

$$\text{spset}(p, \mathcal{D}) = \{\langle h(t), h(s)\rangle \mid h \in H_p \wedge h \models \varphi_p\},$$

i.e., the set of tuple pairs satisfying p. Similarly, given a conjunction X of predicates, we define the *support set* of X as follows:

$$\text{spset}(X, \mathcal{D}) = \{\langle h(t), h(s)\rangle \mid \forall p \in X(\langle h(t), h(s)\rangle \in \text{spset}(p, \mathcal{D}))\},$$

i.e., the set of all tuple pairs satisfying *all* predicates in X.

Given $\varphi: X \to p_0$, assume that H is the set of all valuations of φ in $\mathcal{D}$, and t_0 and s_0 are the tuple variables used in p_0. Then the *support set* of φ, denoted by spset$(\varphi, \mathcal{D})$, is defined as

$$\text{spset}(\varphi, \mathcal{D}) = \{\langle h(t_0), h(s_0)\rangle \mid h \in H \wedge h \models X \wedge h \models \varphi\}.$$

To quantify the frequency of φ, we define the *support* of φ as

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

Similarly we define the notions of supp$(p, \mathcal{D})$ and supp$(X, \mathcal{D})$.

For an integer σ, an REE is *σ-frequent* on $\mathcal{D}$ if supp$(\varphi, \mathcal{D}) \geq \sigma$.

Theorem 1: *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$(\varphi, \mathcal{D})$. □

Proof. There are two cases to consider: (1) φ and φ' use the same set of tuple variables. In this case, spset$(\varphi', \mathcal{D})$ is clearly a subset of spset$(\varphi, \mathcal{D})$, since the predicates that those tuple variables have to satisfy in φ make a subset of those in φ', and hence more valuations can contribute to the support of φ. (2) REE φ' uses more tuple variables than φ. By the definition of support, the additional tuple variables used in φ' will not increase the support. Indeed, for each $\langle h(t_0), h(s_0)\rangle$ in spset$(\varphi', \mathcal{D})$, $\langle h(t_0), h(s_0)\rangle$ must also be in spset$(\varphi, \mathcal{D})$, since otherwise h cannot satisfy φ'. In both cases, spset$(\varphi', \mathcal{D}) \subseteq$ spset$(\varphi, \mathcal{D})$ and thus supp$(\varphi', \mathcal{D}) \leq$ supp$(\varphi, \mathcal{D})$. □

Example 4: In Example 3, $\varphi \preceq \varphi'$. By Theorem 1, supp$(\varphi, \mathcal{D}) \geq$ supp$(\varphi', \mathcal{D})$, since spset$(\varphi, \mathcal{D})$ = spset$(\varphi', \mathcal{D}) = \{t_3 \mapsto t_a\}$. □

Confidence. Confidence indicates how often an REE $\varphi: X \to p_0$ has been found true, given that X is satisfied.

For an REE $\varphi: X \to p_0$, the *confidence* of φ on $\mathcal{D}$ denoted by conf$(\varphi, \mathcal{D})$, is defined to be conf$(\varphi, \mathcal{D}) = \frac{\text{supp}(X \wedge p_0, \mathcal{D})}{\text{supp}(X, \mathcal{D})}$.

For a threshold δ, an REE is *δ-confident* on $\mathcal{D}$ if conf$(\varphi, \mathcal{D}) \geq \delta$. The use of confidence helps us tolerate noise, such that useful rules could still be discovered from the noisy data.

Besides support and confidence, we also use *attribute diversity*, *minimality* and *succinctness* as objective measures, which are presented in [1] for the lack of space. Note that other objective measures can be plugged in our interestingness model when needed.

3.1.2 Subjective Measures. Different users have diverse preference for rules. The assumption that users have a universal preference is not always grounded. Below we train an ML model $\mathcal{M}_{\text{sub}}$ for capturing subjective user preference, to compensate objective ones.

Model architecture. We learn ML model $\mathcal{M}_{\text{sub}}$ by first transforming each rule into an embedding, and then feeding it into a lightweight model, which outputs a scalar score, indicating the subjective preference of users. The details are explained as follows.

Embedding. Given an REE $\varphi: X \to p_0$, we create a rule embedding. We learn the embedding in a hierarchical manner such that token embeddings, predicate embeddings and rule embeddings are generated one after another, and finally the subjective score is computed.

(1) Firstly, we embed each predicate p in X by tokenizing its operator and operands into three tokens, say T_1, T_2, and T_3: (1) if p is $t.A \oplus c$, we tokenize it as $t.A$, $\oplus$ and c; similarly for $t.A \oplus s.B$; and (2) if p is an ML predicate $\mathcal{M}(t[\bar{A}], s.[\bar{B}])$, we treat it as $\mathcal{M}()$, $t[\bar{A}]$ and

$s[\bar{B}]$. Similar to NLP tasks, we treat tokens as words and construct a vocabulary $\mathcal{V}$. For each $T_i \in \mathcal{V}$, we use ELMo [56], a state-of-the-art word representation model, to transform T_i to a d-dimensional representation $\mathbf{T}_i \in \mathbb{R}^{d \times 1}$. Then we adopt a linear layer $\mathbf{w}_{\text{emb}} \in \mathbb{R}^{3 \times 1}$ to generate the predicate embedding $E_p \in \mathbb{R}^{d \times 1}$ of p:

$$E_p = [\mathbf{T}_1; \mathbf{T}_2; \mathbf{T}_3]\mathbf{w}_{\text{emb}},$$

where $[;]$ denotes concatenation; $\mathbf{w}_{\text{emb}}$ is shared by all predicates.

(2) After obtaining the predicate embeddings for each p in X, we compute the *precondition embedding*, denoted by $E_X \in \mathbb{R}^{d_r \times 1}$ for X. Recall that X is a conjunction $p_1 \wedge p_2 \wedge \cdots \wedge p_{|X|}$ of predicates. The embedding E_X should satisfy the *permutation invariant* property, *i.e.,* $E_X = E_{X'}$, where $X' = p_{\pi_1} \wedge \cdots \wedge p_{\pi_{|X|}}$, and $\pi_1, \ldots, \pi_{|X|}$ is a new permutation of $1, \ldots, |X|$. To achieve the permutation invariant property, we adopt *deep sets* [77] based on the predicate embeddings, and obtain $E_X = \rho(\sum_{i=1}^{|X|} \Phi(E_{p_i}))$, where ρ and Φ are two linear layers without activation functions. Moreover, we use predicate embedding of p_0 as the consequence embedding, denoted by E_{p_0}.

(3) Finally, the precondition embedding E_X and the consequence embedding E_{p_0} are concatenated together to form the rule embedding, denoted by $E_\varphi \in \mathbb{R}^{(d_r+d) \times 1}$, as shown in Equation 1:

$$E_\varphi = [E_X^{\mathsf{T}}; E_{p_0}^{\mathsf{T}}]^{\mathsf{T}}. \tag{1}$$

Lightweight model. Given the rule embedding E_φ, our lightweight model employs a fully-connected layer with the Sigmoid activation function to output the final subjective interestingness score:

$$\mathcal{M}_{\text{sub}}(\varphi) = \text{Sigmoid}(\mathbf{w}_{\text{light}}^{\mathsf{T}} E_\varphi + b_{\text{light}}),$$

where $\mathbf{w}_{\text{light}} \in \mathbb{R}^{(d_r+d) \times 1}$ and $b_{\text{light}} \in \mathbb{R}$ are parameters of the lightweight model. Since we adopt the Sigmoid activation function, the subjective score is guaranteed to be in the range $[0, 1]$,

Example 5: Consider φ_1 in Example 1. We first tokenize each of its predicates, *e.g.,* $\{\mathcal{M}_{\text{addr}}, t_a.\text{address}, t_b.\text{address}\}$ of $p = \mathcal{M}_{\text{addr}}(t_a.\text{address}, t_b.\text{address})$, and transform them into embeddings using ELMo. The representation of p is generated using $\mathbf{w}_{\text{emb}}$. After all predicates are embedded, compute the precondition and consequence embeddings of φ_1, and concatenate them to form E_{φ_1}. Then the subjective score is computed via $\mathcal{M}_{\text{sub}}$. □

Remark. Note that we do not adopt existing language models to acquire the rule-level representations for training $\mathcal{M}_{\text{sub}}$, for the following reasons: (1) REEs do not follow natural language structure and thus, directly applying language model (*e.g.,* Bert [14]) does not work well; (2) in REEs, token segmentation possesses different characteristic as text; and (3) given $\varphi : X \rightarrow p_0$, X is a conjunction of predicates, but not a sequence of predicates as in text for which there is no guarantee for the permutation invariant property.

3.2 Modeling User Preference

Putting these together, we are ready to define the *interestingness* of an REE φ. Let F and G be the set of objective and subjective measures. The interestingness of φ is defined to be

$$\text{interest}(\varphi) = \sum_{f \in F} w_f f(\varphi) + \sum_{g \in G} w_g g(\varphi),$$

where w_f and w_g are non-negative weights associated with each measures and they are sum up to 1 (we apply Softmax on learned parameters to guarantee this property). Moreover, we assume *w.l.o.g.* that each measure f in F outputs a scalar value from 0 to 1, where a larger value is more preferable; similarly for G.

Denote the complete weight vector by $\mathbf{w}_{\text{interest}}$, representing the user preference. The weight vector *depends on users' preference.* Intuitively, a larger weight indicates that the corresponding measurement is more important to the user. Once the weight vector is determined, we can compute the interestingness of each rule, based on which we can deduce the top-k interesting rules.

As will be seen in Section 3.3, all parameters (including weight vector and subjective measures) are *learned.* Users may also manually adjust $\mathbf{w}_{\text{interest}}$. Some users might rank support the most important, while the others give confidence a higher priority. Domain experts may want to find unexpected rules to catch critical errors, which cannot be detected by the obvious rules that they have accumulated; hence they can assign a larger weight to $\mathcal{M}_{\text{sub}}$. Novice users can turn off the subjective features by setting $w_g = 0$, and only use objective measures to find common rules as mined by conventional rule discovery algorithms. If users do not know exactly how to rank the rules, they could label a few rules, and our model (esp. G) can learn their preference. This highlights the need for learning individualistic preference by interacting with users.

As a real-life example, our biopharmaceutical users have accumulated quite a few rules for identifying drugs for a disease, and the conventional support and confidence do not help them discover new rules. What they need is the subjective measure (*i.e.,* G); hence they set w_f very small and let w_g dominate in $\mathbf{w}_{\text{interest}}$. The subjective measure$\mathcal{M}_{\text{sub}}$ here suffices since it is learned via a neural network that has enough approximation power for users' preference.

3.3 An Active Learning Approach

Denote the combination of $\mathcal{M}_{\text{sub}}$ and $\mathbf{w}_{\text{interest}}$ by $\mathcal{M}_{\text{interest}}$. Below we show how to jointly learn $\mathcal{M}_{\text{interest}}$ based on *active learning.*

We adopt a pairwise ranking setting for users to label the partial orders of a few rules, since it is impractical to ask the users to label the actual interestingness score interest(φ) for each individual rule φ. More specifically, we maintain a rule pool $\mathcal{S}_{\text{REEs}}$. Given a pair of rules $\langle \varphi_i, \varphi_j \rangle$ in $\mathcal{S}_{\text{REEs}}$, a user may label 1 on $\langle \varphi_i, \varphi_j \rangle$ if s/he thinks that φ_i is more interesting than φ_j, denoted by $\varphi_j \ll \varphi_i$; otherwise, the user labels the pair 0. We denote the label by $y^{(i,j)} \in \{0, 1\}$. For each training instance $\langle \varphi_i, \varphi_j, y^{(i,j)} \rangle$, we adopt the Siamese neural network with shared parameters to separately compute the interestingness scores of φ_i and φ_j. Then we use the Cross Entropy loss function to train $\mathcal{M}_{\text{interest}}$ as follows.

$$\Pr(\varphi_i \ll \varphi_j) = \text{Sigmoid}(\text{interest}(\varphi_i) - \text{interest}(\varphi_j))$$

$$\mathcal{L}_{\text{CE}} = \sum_{i,j} y^{(i,j)} \log(\Pr(\varphi_i \ll \varphi_j)) + (1 - y^{(i,j)}) \log(1 - \Pr(\varphi_i \ll \varphi_j)).$$

Active learning. Since there are $\frac{1}{2}|\mathcal{S}_{\text{REEs}}| \times |\mathcal{S}_{\text{REEs}}|$ pairs of rules, it is impractical for users to label all of them. Thus we adopt an active learning approach, which selects high-quality pairs of rules for users to label. We iteratively learn $\mathcal{M}_{\text{interest}}$ and actively select rule pairs in $\mathcal{S}_{\text{REEs}}$ that $\mathcal{M}_{\text{interest}}$ cannot distinguish well, *i.e.,* pairs that have the smallest differences in interestingness scores, and ask users for labeling. The process proceeds until either it reaches the maximum number of iterations or the accuracy of $\mathcal{M}_{\text{interest}}$ reaches a predefined bound in a validation data. In the training step, $\mathcal{M}_{\text{interest}}$ only focuses on learning rules' interestingness; thus we could simply combine different predicates and generate as many rules as required for $\mathcal{S}_{\text{REEs}}$. We defer the details to [1] for the lack of space. We find that it often suffices for users to label 50 rule pairs in 5 rounds of interaction. After training, $\mathcal{M}_{\text{interest}}$ is used to rank *valid* rules in the discovery process, as will be seen in Section 4.

4 INTERESTING RULE DISCOVERY

Based on the interestingness model, we discover top-k interesting rules (Section 4.1) and develop such an algorithm (Section 4.2).

4.1 Problem Statement

Denote by Σ_{all} the set of minimal REEs on $\mathcal{D}$ that are σ-frequent and δ-confident for thresholds σ and δ. As remarked earlier, Σ_{all} may contain an excessive number of rules that are not very relevant to users' applications and interest. To reduce such rules, we adopt the following strategies. (1) We pick an application-dependent set of candidate consequences p_0, denoted by RHS, which pertain to a particular application of users. (2) For each p_0 in RHS, we learn a *subset* $\mathcal{P}_0$ of predicates, including all predicates correlated to p_0; we only focus on discovering REEs φ: $X \longrightarrow p_0$ such that $X \subseteq \mathcal{P}_0$.

In practice, users are typically familiar with their datasets and do not find it hard to set RHS. A real-life case is to find duplicate tuples Organization, and users adopt RHS of the form t_0.id = t_1.id. To fill in missing values in the attribute Country, they simply chose predicates of RHS related to Country. The correlated attributes can also be identified by using, *e.g.,* LSTM networks [?].

Top-k REEs discovery. The *top-k discovery* problem is as follows.

- *Input*: A schema $\mathcal{R}$, an instance $\mathcal{D}$ of $\mathcal{R}$, a consequence set RHS, the support/confidence thresholds σ/δ, an integer k, and interestingness model interest with learned weight vector.
- *Output*: A set Σ consisting of the top-k interesting REEs *w.r.t.* $\mathbf{w}$ such that for each $\varphi : X \longrightarrow p_0$ in Σ, (a) $p_0 \in$ RHS; (b) $X \subseteq \mathcal{P}_0$, where $\mathcal{P}_0$ is a set of predicates correlated to p_0; and (c) φ is minimal and moreover, it is σ-frequent and δ-confident.

All the discovered rules are valid and reliable as ensured by support and confidence, along the same lines as [55].

4.2 A Top-k Discovery Algorithm

We start with *pruning strategies* to remove early those REEs that are unlikely to become top-k interesting rules, and reduce the large number of candidate rules to be examined in top-k rule discovery.

Pruning strategy. Our strategies are based on *anti-monotonicity* and *interestingness* and thus, REEs with high orders **[P1]**, low supports **[P2]** or low interestingness **[P3]** are pruned early.

We maintain a heap Σ of top-k minimal REEs that are discovered so far. Denote the k-th highest interestingness of rules in Σ by T_k. Assume that we are checking whether a candidate REE $\varphi : X \longrightarrow p_0$ is one of the top-k interesting rules; if not, we revise and expand X with more predicates from $\mathcal{P}_0$ to make such an REE if possible.

[P1] Prune non-minimal REEs: Before we perform exact checking for φ, we first check whether there exists an REE φ' discovered so far such that $\varphi' \preceq \varphi$. If so, we can skip the processing and expansion for φ, since φ and all of its subsequent expansions are not minimal.

[P2] Prune REEs *with low support*: If supp$(\varphi, \mathcal{D}) < \sigma$, we do not consider any φ' such that $\varphi \preceq \varphi'$ since by Theorem 1, we have that supp$(\varphi', \mathcal{D}) \leq$ supp$(\varphi, \mathcal{D}) < \sigma$ and thus, φ' is not σ-frequent.

[P3] Prune non-interesting REEs: We maintain T_k, the k-th highest interestingness in Σ. We compute an interestingness upper bound UB for rules expanded from φ. If the upper bound is less than T_k, we know that no rules expanded from φ can make a top-k interesting rule and thus, we stop the exact expansion immediately.

We compute UB by taking the minimum of (a) exact bound that safely prunes non-interesting REEs and (b) a learned bound via an ML model. We next show how to find exact and learned bounds.

(1) Exact bound. We categorize the interestingness measures into three types: *monotonic, anti-monotonic and general measures*, based on which we develop the interestingness upper bound. (a) An interesting measure h (including $f \in F$ and $g \in G$) is *monotonic* if $h(\varphi) \leq h(\varphi')$ as long as $\varphi \preceq \varphi'$, *i.e.,* adding predicates to φ monotonically increases the interestingness. Then the upper bound (UB) of $h(\varphi')$ is $h(\mathcal{P}_0 \longrightarrow p_0)$. (b) Adding predicates in an *anti-monotonic* measure monotonically decreases the interestingness, *e.g.,* support. Here the UB of $h(\varphi')$ is $h(\varphi)$. (c) If an interesting measure does not have the above properties, it is referred to as *general, e.g.,* usually the subjective model. The UB of $h(\varphi) = 1$ (see [1] for details).

Example 6: Assume that the k-highest interestingness T_k in Σ is 0.8, and we are currently processing φ. If the exact UB of the REEs expanded from φ is found to be 0.7, we can stop the expansion of φ immediately, since it will not yield any top-k rules. □

(2) Learned bound. We develop an ML model to learn the upper bound, especially for the *general* measures, before the discovery process. Given the interestingness measure $\mathcal{M}_{\text{interest}}$ and a potential REE $\varphi : X \longrightarrow p_0$, we learn a function UBSCORE such that UBSCORE$(\varphi) \approx \max\{\mathcal{M}_{\text{interest}}(\varphi') \mid \varphi' : X \cup P' \longrightarrow p_0, \forall P' \subseteq \mathcal{P}_0\}$.

More specifically, we utilize Deep Q-learning (DQN) [50] to learn UBSCORE. It takes the currently selected $\mathcal{P}_{\text{sel}}$ as state s, and the added predicate p as action a. In each step of one episode, the agent interacts with the environment, *i.e.,* $\mathcal{M}_{\text{interest}}$, to compute an reward $r = \mathcal{M}_{\text{interest}}(\mathcal{P}_{\text{sel}} \cup \{p_0\} \longrightarrow p_0) - \mathcal{M}_{\text{interest}}(\mathcal{P}_{\text{sel}} \longrightarrow p_0)$. DQN contains two networks: a Q-network and a target network. The Q-network takes a state s and an action a as input, and outputs the reward of taking a. It is learned and updated in each state from s to s', where $s' = \mathcal{P}_{\text{sel}} \cup \{p\}$. The Q value is computed as

$$Q(s', p') = \mathbb{E}_{s \sim \xi}[r + \gamma \max_p Q(s, p) | s', p'], \tag{2}$$

where ξ is $\mathcal{M}_{\text{interest}}$, and γ is a discount ratio. Then the Q value is approximated by the Q-network, and the target network is obtained by cloning Q-network within a few steps. We denote the state s (*i.e.,* $\mathcal{P}_{\text{sel}}$) as a $|\mathcal{P}_0|$-dimensional bit vector $\mathbf{v}_s$, where $\mathbf{v}_s[p] = 1$ if $p \in \mathcal{P}_{\text{sel}}$, and $\mathbf{v}_s[p] = 0$ otherwise. We implement the Q-network as a feed-forward network (FFN) and output a $|\mathcal{P}_0|$-dimensional vector $\mathbf{v}_o$ of $|\mathcal{P}_0|$ rewards, such that $\mathbf{v}_o[p]$ is the reward of $\langle \mathcal{P}_{\text{sel}}, p \rangle$:

$$\text{UBSCORE}(\mathcal{P}_{\text{sel}}, p) = \text{Q−network}(\mathbf{v}_s)[p]. \tag{3}$$

By the max in Equation 2, UBSCORE is the upper bound of $\langle \mathcal{P}_{\text{sel}}, p \rangle$. The learning method and loss function are the same as DQN. We adopt the Double and Dueling strategies [33] to speed up training.

Before rule discovery, we train DQN by sampling a few REEs and get UBSCORE, which is then used in rule discovery for pruning.

The training does not dominate the discovery time because (1) The cost of computing rewards is small in $O(|\mathcal{M}_{\text{interest}}|)$ time; (2) Q-network is implemented as an FFN with only a few hidden layers; and (3) the training episode is a constant value, *e.g.,* 2000.

Algorithm. We now present our algorithm, referred to as Topk-Miner, for top-k interesting REEs discovery on dataset $\mathcal{D}$.

As shown in Figure 1, Topk-Miner is a *levelwise search* algorithm. It first initializes a max-heap Σ of maximum size k (line 1), which is used to store the top-k REEs discovered so far, ordered by their interestingness. Given a consequence p_0 in RHS and its correlated $\mathcal{P}_0$, we maintain two predicate sets for discovering new REEs $\varphi : X \rightarrow p_0$ with $X \subseteq \mathcal{P}_0$: (1) $\mathcal{P}_{\text{sel}}$, the set of predicates selected to constitute X; and (2) $\mathcal{P}_{\text{re}}$, the set of remaining predicates in $\mathcal{P}_0$. Initially, $\mathcal{P}_{\text{sel}}$ is empty and $\mathcal{P}_{\text{re}}$ is $\mathcal{P}_0$ (line 4). Topk-Miner then traverses the search space level by level by maintaining a queue Q (line 7), where at the i-th level, it discovers $\varphi : X \rightarrow p_0$ with $|X| = i$. It iteratively adds predicates from $\mathcal{P}_{\text{re}}$ to $\mathcal{P}_{\text{sel}}$ (line 18-19) until one of the following conditions is satisfied: (1) $\mathcal{P}_{\text{re}}$ is exhaustive; or (2) $\varphi : \mathcal{P}_{\text{sel}} \rightarrow p_0$ is a minimal REE (line 10-14), since in this case, adding more predicates will not make $\text{supp}(\varphi, \mathcal{D})$ larger, while it increases the order of φ. We maintain the k-th highest interestingness T_k of REEs in Σ (line 10). If $\text{interest}(\varphi) > \mathsf{T}_k$, Σ is updated (line 12-13) by adding φ into Σ and removing the least interesting REE from Σ if there are more than k REEs in Σ. If $\varphi : \mathcal{P}_{\text{sel}} \rightarrow p_0$ is still not a minimal REE, we expand it (line 18-19); before expansion, we apply the pruning strategies **[P1]-[P3]** (line 15-17), to check whether we can terminate the current expansion early.

Algorithm Topk-Miner employs the following optimization strategies commonly used in rule discovery. (a) When multiple p_0 in RHS share similar correlated predicates $\mathcal{P}_0$, it processes these p_0's together (not shown). (b) It pre-computes auxiliary structures (line 2) such as position list indexes (PLI) [55] to efficiently compute supports and confidences for REEs verification.

We also propose the following for top-k REE discovery.

Correlated predicate learning. Recall that for each consequence p_0, we learn a subset $\mathcal{P}_0$ of its correlated predicates, which includes both logic predicates and ML predicates. To this end, we maintain a pool of pre-trained ML models. Given a schema $\mathcal{R}$, we associate attributes in $\mathcal{R}$ to compatible ML models in the pool and initialize the ML predicates. Then, to learn correlated $\mathcal{P}_0$ for a given p_0, we apply graphical lasso [25, 78] to learn how an attribute is affected by other attributes. Informally, given a predicate p, either a logic or an ML predicate, if attributes in p have strong impact on the attributes in p_0, p is correlated to p_0 and is included in $\mathcal{P}_0$.

Handling multiple relation atoms. To efficiently support multiple relation atoms in Topk-Miner, we incrementally discover multi-variable REEs in rounds at each level (not shown). In the j-th round, we discover j-variable REEs by processing each non-minimal $(j-1)$-variable REE φ found in the $(j-1)$-th round, by constructing $\mathcal{P}_{\text{re}}$, which is the set of remaining predicates that can be used to expand φ, such that the expanded rules contain exactly j relation atoms. Here $\mathcal{P}_{\text{re}}$ is built incrementally by enumerating the predicates that contain one new relation atom and at most one existing relation atom used in φ. Then we discover j-variable rules by expanding φ with the predicates in the newly constructed $\mathcal{P}_{\text{re}}$.

```
Algorithm Topk-Miner
Input: $\mathcal{D}$, RHS, $k$, $\sigma$ and $\delta$.
Output: A heap $\Sigma$ of top-$k$ interesting REEs such that for each $\varphi : X \rightarrow p_0$ in $\Sigma$,
    (1) $p_0 \in$ RHS; (2) $X \subseteq \mathcal{P}_0$, where $\mathcal{P}_0$ is a set of predicates correlated to $p_0$.
1.  $\Sigma$ := an empty max-heap of maximum size $k$, ordered by rule interestingness;
2.  Build auxiliary structures, e.g., position list indexes (PLI) [55];
3.  for each $p_0 \in$ RHS do
4.      $\mathcal{P}_{\text{sel}} := \emptyset$; $\mathcal{P}_{\text{re}} := \mathcal{P}_0$;
5.      $\Sigma := \text{Expand}(\mathcal{D}, \mathcal{P}_{\text{sel}}, \mathcal{P}_{\text{re}}, p_0, k, \delta, \sigma, \Sigma)$;
6.  return $\Sigma$;

Procedure Expand
Input: $\mathcal{D}$, $\mathcal{P}_{\text{sel}}$, $\mathcal{P}_{\text{re}}$, $p_0$, $k$, $\delta$, $\sigma$ and the current heap $\Sigma$ of interesting REEs.
Output: An updated heap $\Sigma$ of interesting REEs.
7.  $Q$ := an empty queue; $Q$.add($\langle \mathcal{P}_{\text{sel}}, \mathcal{P}_{\text{re}} \rangle$);
8.  while $Q \neq \emptyset$ do
9.      $\langle \mathcal{P}_{\text{sel}}, \mathcal{P}_{\text{re}} \rangle := Q$.pop();
10.     $\varphi := \mathcal{P}_{\text{sel}} \rightarrow p_0$; $\mathsf{T}_k$ := the $k$-th highest interestingness in $\Sigma$;
11.     if $\varphi$ is minimal (and thus, it is $\sigma$-frequent and $\delta$-confident) then
12.         if interest($\varphi$) $> \mathsf{T}_k$ then
13.             Update $\Sigma$ using $\varphi$;
14.         continue;
15.     UB := min (exact bound, UBSCORE bound) of rules expanded from $\varphi$;
16.     if $\exists \varphi' \in \Sigma$ s.t. $\varphi' \preceq \varphi$ [P1] or supp($\varphi$) $< \sigma$ [P2] or UB $< \mathsf{T}_k$ [P3]
17.         continue; // Early termination of the current expansion
18.     for each $p \in \mathcal{P}_{\text{re}}$ do // Add predicates from $\mathcal{P}_{\text{re}}$ to $\mathcal{P}_{\text{sel}}$
19.         $Q$.add($\langle \mathcal{P}_{\text{sel}} \cup \{p\}, \mathcal{P}_{\text{re}} \setminus \{p\} \rangle$)
20. return $\Sigma$;
```

Figure 1: Algorithm Topk-Miner

Early termination. Topk-Miner allows early termination for top-k discovery, *i.e.,* we can terminate the expansion of an REE φ early if one of the following happens (line 13): (1) if there exists an REE φ' in Σ such that $\varphi' \preceq \varphi$, then there is no need to expand φ, which cannot be minimal **[P1]**; (2) if $\text{supp}(\varphi) \leq \sigma$ **[P2]**, then further expanding φ will not lead to σ-frequent REEs by the anti-monotonicity of support; and (3) if UB $< \mathsf{T}_k$ **[P3]**, where UB $= \min\{\text{exact bound}, \text{learned bound}\}$ is the interestingness upper bound of the rules expanded from φ, then no rule expanded from φ is more interesting than any one in Σ, and we can stop immediately.

We further optimize Topk-Miner by considering effective processing orders for the predicates in $\mathcal{P}_{\text{re}}$ (line 18). Below we present two processing orders of $\mathcal{P}_{\text{re}}$ based on support and interestingness.

(1) Support-based processing order. The first order is to add the predicates p from $\mathcal{P}_{\text{re}}$ to $\mathcal{P}_{\text{sel}}$ based on $\text{supp}(\mathcal{P}_{\text{sel}} \wedge p, \mathcal{D})$, such that predicates with high supports are processed first. This helps us prune those predicates in $\mathcal{P}_{\text{re}}$ that are useless in generating σ-frequent REEs (in addition to **[P2]**). Intuitively, if including a predicate p with high support cannot lead to an σ-frequent REE, it is even more difficult for a predicate p' with low support to do so. Formally,

Lemma 2: *Given an* REE $\varphi : \mathcal{P}_{\text{sel}} \rightarrow p_0$*, and two predicates* p *and* p' *in* $\mathcal{P}_{\text{re}}$*, expanding* $\mathcal{P}_{\text{sel}}$ *with* p' *will not give any* σ*-frequent* REE *if* $\mathcal{P}_{\text{sel}} \wedge p \rightarrow p_0$ *is not* σ*-frequent and* $\text{spset}(p', \mathcal{D}) \subseteq \text{spset}(p, \mathcal{D})$. □

Proof. Let φ and φ' be $\mathcal{P}_{\text{sel}} \wedge p \rightarrow p_0$ and $\mathcal{P}_{\text{sel}} \wedge p' \rightarrow p_0$, respectively. Since $\text{spset}(p', \mathcal{D}) \subseteq \text{spset}(p, \mathcal{D})$, we have $\text{spset}(\mathcal{P}_{\text{sel}} \wedge p' \wedge p_0, \mathcal{D}) \subseteq \text{spset}(\mathcal{P}_{\text{sel}} \wedge p \wedge p_0, \mathcal{D})$. Then for each h' in $\text{spset}(\varphi', \mathcal{D})$, h' must be in $\text{spset}(\varphi, \mathcal{D})$. Thus, $\text{supp}(\varphi', \mathcal{D}) \leq \text{supp}(\varphi, \mathcal{D}) < \sigma$. □

That is, if $\mathcal{P}_{\mathrm{re}}$ is ordered by supports, every time we see a predicate p and if $\mathcal{P}_{\mathrm{sel}} \wedge p \longrightarrow p_0$ is not σ-frequent, we can prune all p' in $\mathcal{P}_{\mathrm{re}}$ ordered after p with $\mathrm{spset}(p', \mathcal{D}) \subseteq \mathrm{spset}(p, \mathcal{D})$.

Example 7: Consider the relation in Table 2. Assume that $\mathcal{P}_{\mathrm{sel}} = \{\mathrm{RentalContract(t)}\}$ and p_0 is $t.\mathrm{montly\_rent} = 1,500$. Let p and p' be $t.\mathrm{renter}$ = "Ada" and $t.\mathrm{pid} = p_3$, respectively. Clearly, $\mathrm{spset}(p', \mathcal{D}) = \{t_7 \mapsto t, t_8 \mapsto t\}$ is a subset of $\mathrm{spset}(p, \mathcal{D}) = \{t_7 \mapsto t, t_8 \mapsto t, t_9 \mapsto t\}$ and thus, p is processed before p'. If we find that $\mathcal{P}_{\mathrm{sel}} \wedge p \longrightarrow p_0$ is not σ-frequent, there is no need to process p', since $\mathrm{supp}(\mathcal{P}_{\mathrm{sel}} \wedge p \longrightarrow p_0) = 2 > 1 = \mathrm{supp}(\mathcal{P}_{\mathrm{sel}} \wedge p' \longrightarrow p_0)$. □

(2) Interestingness-based processing order. Alternatively, we can also process the predicates in $\mathcal{P}_{\mathrm{re}}$ based on their "potential" in discovering rules with high interestingness. Intuitively, if more high interestingness rule are discovered, the interestingness bound T_k (the k-th highest interestingness observed so far) is tighter and thus, more rules are likely to be pruned by **[P3]** at an earlier stage.

More specifically, given REE $\varphi : \mathcal{P}_{\mathrm{sel}} \longrightarrow p_0$ and a predicate p in $\mathcal{P}_{\mathrm{re}}$, we define an indicator Δ_p, expressing the best possible interestingness that can be achieved by including p to $\mathcal{P}_{\mathrm{sel}}$:

$$\Delta_p = \min\left\{ \sum_{f \in F} w_f f_{\mathrm{ub}}(\varphi') + \sum_{g \in G} w_g g_{\mathrm{ub}}(\varphi'), \mathrm{UBSCORE}(\varphi') \right\},$$

where φ' is $\mathcal{P}_{\mathrm{sel}} \wedge p \longrightarrow p_0$, and $f_{\mathrm{ub}}, g_{\mathrm{ub}}$ are exact bounds.

Then, the predicate p in $\mathcal{P}_{\mathrm{re}}$ with the maximum Δ_p is selected as the next predicate to be used to expand $\mathcal{P}_{\mathrm{sel}}$.

Remark. The support and interestingness based processing orders can be combined together, *e.g.,* by ordering predicates in $\mathcal{P}_{\mathrm{re}}$ by their supports and breaking ties based on the interestingness indicator.

Example 8: We show how φ_2 in Example 1 is found. Assume that $\mathcal{P}_{\mathrm{sel}} = \{\mathrm{Property}(t_a), \mathrm{RentalContract}(s_a), \mathrm{RentalContract}(s_b), \mathcal{M}_{\mathrm{rent}}(t_a, s_a)\}$, $\mathcal{P}_{\mathrm{re}} = \{t_a.\mathrm{pid} = s_a.\mathrm{pid}, s_a.\mathrm{rent\_start\_year} \neq s_b.\mathrm{rent\_start\_year}\}$ and p_0 is $s_a.\mathrm{monthly\_rent} \neq s_b.\mathrm{monthly\_rent}$. We expand $\mathcal{P}_{\mathrm{sel}}$ by adding predicates in $\mathcal{P}_{\mathrm{re}}$ one by one. Before we perform the exact expansion, we first compute an interestingness upper bound, say UB_l. If UB_l is less than T_k, we can terminate early since expanding $\mathcal{P}_{\mathrm{sel}}$ will not result in any top-k interesting REEs **[P3]**. Otherwise, assume that $p : t_a.\mathrm{pid} = s_a.\mathrm{pid}$ has a larger $\mathrm{supp}(\mathcal{P}_{\mathrm{sel}} \wedge p)$. Then we add p to $\mathcal{P}_{\mathrm{sel}}$ first, by the support-based processing order, until we find that φ_2 is a minimal REE. If φ_2 is more interesting than some rules in Σ, Σ is updated using φ_2. □

Correctness. The correctness of Topk-Miner is verified as follows.

Theorem 3: Topk-Miner *correctly discovers top-k* REEs. □

Proof. One can verify that Topk-Miner does not miss any rule φ unless there exists an REE φ' with $\varphi' \preceq \varphi$, such that when processing φ', we decide to stop expanding φ' according to the pruning strategies **[P1]-[P3]**. This is correct since by **[P1]-[P3]**, φ cannot be a top-k REE, since it is non-minimal, not σ-frequent, or less interesting than at least k rules that have been discovered. □

5 ANYTIME DISCOVERY

In this section, we extend Topk-Miner and develop an anytime algorithm Anytime-Miner, such that users can continuously get the next top-k interesting rules when needed, via lazy evaluation.

A brute-force approach for supporting this is to compute the full ranking of all REEs first. Every time a user wants the next top-k results, we retrieve the corresponding results from the ranked list. Clearly, this method is inefficient. Users are typically only interested in the first few top-ranked results, and should not pay the cost of waiting for discovering the entire set of REEs on a dataset.

We convert Topk-Miner into Anytime-Miner. Denote by Σ (see Figure 1) the heap of discovered top-ranked REEs so far. We expand Σ to *lazily* discover the next top-k results as follows.

(1) Instead of just maintaining the top-k results in Σ, all minimal rules discovered are kept in Σ, referred to as *complete rules.* In addition, we maintain *partial rules* in Σ, where an REE φ is said to be *partial* if at the time it is processed, its interestingness upper bound is lower than those of at least k complete rules in Σ. In other words, the pruning strategy **[P3]** in the original Topk-Miner is revised: instead of directly dropping those rules with relatively low interestingness, we keep them as partial rules in Σ. Intuitively, these partial rules are likely to be expanded and contribute to the top-k ones in later rounds when the user wants more results. For partial rules, their remaining predicates, say $\mathcal{P}_{\mathrm{re}}$, are also stored for later expansion, and we use its interestingness upper bound as the key in Σ.

(2) We only return the next top-k *complete rules* in Σ. For each partial rule whose interestingness upper bound is among the next top-k, we resume its levelwise search in order. Since we have stored the remaining predicates $\mathcal{P}_{\mathrm{re}}$ for each partial rule, the resumption is straightforward. The resumed search updates the rules maintained in Σ; it continues until the next top-k rules in Σ all become complete.

(3) We ensure that the next top-k results are not "redundant", *i.e.,* not logical consequences of the rules that have been shown before. Thus, we apply the implication analysis [21] on each newly discovered rules. Formally, we say that a set of REEs Σ *entails* another REE φ over $\mathcal{R}$, denoted by $\Sigma \models \varphi$, if for any instance $\mathcal{D}$ of $\mathcal{R}$, if $\mathcal{D} \models \Sigma$ then $\mathcal{D} \models \varphi$. Then, every time a complete rule φ is discovered, we add it to the heap Σ only if $\Sigma \not\models \varphi$. While the implication problem is Π_2^p-complete [22], we develop an efficient heuristic for checking.

Example 9: Assume that $k = 3$ and rules in Σ are currently stored in order: $\varphi_1^c, \varphi_2^p, \varphi_3^p, \varphi_4^c$, where φ_i^c and φ_j^p denote complete rules and partial rules, respectively. Since there are partial rules in top-3 of Σ, we process them in order. Assume that we first resume the levelwise search for φ_2^p and obtain three new REEs: $\varphi_5^c, \varphi_6^c, \varphi_7^p$, and Σ is updated as: $\varphi_1^c, \varphi_5^c, \varphi_6^c, \varphi_7^p, \varphi_3^p, \varphi_4^c$. At this point, all top-3 rules in Σ $(\varphi_1^c, \varphi_5^c, \varphi_6^c)$ are complete rules, and they are returned to the user. □

6 PARALLEL TOP-k RULE DISCOVERY

In this section we parallelize top-k discovery to scale with large datasets. We first review a criterion for measuring the effectiveness of parallel algorithms (Section 6.1). We then parallelize Topk-Miner, denoted by PTopk-Miner, with the performance guarantees (Section 6.2); Anytime-Miner is parallelized along the same lines.

6.1 Parallel Scalability

We revisit the widely adopted notion of parallel scalability [40].

Assume that $\mathcal{A}$ is a sequential algorithm which, given a dataset $\mathcal{D}$, consequences RHS and thresholds σ and δ for support and confidence, respectively, computes a set Σ of top-k interesting REEs on $\mathcal{D}$. Denote its worst running time as $t(|\mathcal{D}|, |\mathsf{RHS}|, \sigma, \delta)$. We say that a parallel algorithm $\mathcal{A}_p$ is *parallelly scalable relative to* $\mathcal{A}$ if its running time by using n processors can be expressed as:

$$T(|\mathcal{D}|, |\mathsf{RHS}|, \sigma, \delta) = \widetilde{O}(\frac{t(|\mathcal{D}|, |\mathsf{RHS}|, \sigma, \delta)}{n}),$$

where the notation $\widetilde{O}()$ hides $\log(n)$ factors.

Intuitively, parallel scalability guarantees "linear" speedup of $\mathcal{A}_p$ relative to the "yardstick" algorithm $\mathcal{A}$. That is, the more processors are used, the faster $\mathcal{A}_p$ is. Hence $\mathcal{A}_p$ can scale with large databases by adding processors and makes REEs discovery feasible in practice.

6.2 Parallel Algorithm

We next parallelize Topk-Miner to be PTopk-Miner (Figure 2).

Setting. PTopk-Miner runs with one coordinator S_c and n workers $P_1, \ldots, P_n$ under the Bulk Synchronous Parallel (BSP) model [71], where the coordinator is responsible for distributing and balancing workloads, and workers discover rules in parallel. The overall computation is divided into supersteps of a fixed duration.

Overview. Similar to Topk-Miner, the coordinator maintains a max-heap of maximum size k, consisting of the top-ranked REEs discovered so far (line 1). Denote the heap at superstep i by Σ_i, and the k-th highest interestingness in Σ_i by T_k^i. The coordinator first distributes the workloads evenly to all the workers (see below; line 2-5). Then, each worker parallelly processes its workload and discovers rules in supersteps (line 6-15). At each superstep, the coordinator informs each worker the latest interestingness bound T_k^i (line 10), based on which each worker performs the subsequent discovery (line 11) by applying the pruning strategies in Section 4.2. The coordinator S_c pulls the newly discovered top-k rules from each worker at the end of each superstep (line 12). In addition, it adjusts and balances the workload when needed (line 13-14; see below). Moreover, S_c extends the heap Σ_i to Σ_{i+1} with the new rules, and updates interestingness bound T_k^i to T_k^{i+1} (line 15). The process continues until all workers finish their work, *i.e.,* when no more rules with interestingness above the bound can be found.

Workload assignment. Given RHS, S_c evenly divides it into n partitions, namely $\mathsf{RHS}_1, \ldots, \mathsf{RHS}_n$, and constructs a set of *work units* based on each RHS_j ($j \in [1, n]$) for the j-th worker P_j as follows.

For each consequence p_0 in RHS_j, it constructs a work unit, which is a triple $w = \langle \mathcal{P}_{\mathsf{sel}}, \mathcal{P}_{\mathsf{re}}, p_0 \rangle$, where $\mathcal{P}_{\mathsf{sel}}$ denotes the set of predicates that are selected to constitute the rules, and $\mathcal{P}_{\mathsf{re}}$ denotes the set of remaining predicates. Initially, $\mathcal{P}_{\mathsf{sel}}$ is empty and $\mathcal{P}_{\mathsf{re}}$ is $\mathcal{P}_0$, which is the set of predicates correlated to p_0. Then, it sends workload $\mathcal{W}_j = \{w = \langle \mathcal{P}_{\mathsf{sel}}, \mathcal{P}_{\mathsf{re}}, p_0 \rangle \mid p_0 \in \mathsf{RHS}_j\}$ to worker P_j.

Upon receiving $\mathcal{W}_j$, worker P_j fetches a subset $\mathcal{D}_{\mathcal{W}_j}$ of data from $\mathcal{D}$, guided by $\mathcal{W}_j$, where $\mathcal{D}_{\mathcal{W}_j} = \{t \in \mathcal{D} \mid \exists s \in \mathcal{D}, p \in P_0$ s.t. $h\langle t, s \rangle \models p$ or $h\langle t, s \rangle \models p_0$, where $p_0 \in \mathsf{RHS}_j\}$; it also constructs the corresponding auxiliary structures for performing rule discovery. In this way, the same data will be merged and transmitted to P_j only once even if it satisfies multiple predicates, reducing the total communication cost when processing multiple predicates.

Example 10: Consider relation RentalContract in Table 2. Let p be $t.\mathsf{pid} = s.\mathsf{pid}$ and p' be $t.\mathsf{renter} = s.\mathsf{renter}$. Assume that the coordinator S_c constructs and assigns workload $\mathcal{W}_j = \{w, w'\}$ to worker P_j, where $w = \langle \emptyset, \{p\}, p_0 \rangle$ and $w' = \langle \emptyset, \{p'\}, p_0' \rangle$. It is easy to see that $\mathcal{D}_{\mathcal{W}_j} = \{t_7, t_8, t_9\}$. In particular, although $h\langle t_7, t_8 \rangle \models p$ and $h\langle t_7, t_8 \rangle \models p'$, t_7 and t_8 are only transmitted to P_j once. □

```
Algorithm PTopk-Miner
Input: $\mathcal{D}$, RHS, w, $k$, $\sigma$, $\gamma$, a coordinator $S_c$ and $n$ workers $P_1, \ldots, P_n$.
Output: A max-heap $\Sigma$ of top-$k$ REEs on $\mathcal{D}$.
 /* executed at coordinator $S_c$ */
1.  $i := 0$;  $\Sigma_i :=$ an empty max-heap of maximum size $k$;
2.  for each $p_0 \in$ RHS do
3.      Construct a work unit $w = \langle \mathcal{P}_{\mathsf{sel}}, \mathcal{P}_{\mathsf{re}}, p_0 \rangle$, where $\mathcal{P}_{\mathsf{sel}} = \emptyset$ and $\mathcal{P}_{\mathsf{re}} = \mathcal{P}_0$;
4.  Evenly divide RHS into $n$ partitions, namely $\mathsf{RHS}_1, \ldots, \mathsf{RHS}_n$;
5.  Assign workload $\mathcal{W}_j = \{w = \langle \mathcal{P}_{\mathsf{sel}}, \mathcal{P}_{\mathsf{re}}, p_0 \rangle \mid p_0 \in \mathsf{RHS}_j\}$ to worker $P_j$;
 /* run on n workers in parallel, in supersteps */
6.  for each worker $P_j$ do
7.      Fetch $\mathcal{D}_{\mathcal{W}_j} = \{t \in \mathcal{D} \mid \exists s \in \mathcal{D}, p \in P_0$ s.t. $h\langle t, s \rangle \models p$ or $h\langle s, t \rangle \models p_0$,
            where $p_0 \in \mathsf{RHS}_j\}$ and build the corresponding auxiliary structures;
8.  while there exists unfinished work do   /* superstep i */
9.      for each $P_j$ with non-empty workload $W_j$ do
10.         $\mathsf{T}_k^i :=$ the $k$-th highest interestingness in $\Sigma_i$ (informed by $S_c$);
11.         Run Topk-Miner at $P_j$ based on $\mathcal{W}_j$ and $\mathcal{D}_{\mathcal{W}_j}$ in parallel;
12.         $S_c$ pulls top-$k$ REEs $\varphi$ newly discovered ( interest($\varphi$) $> \mathsf{T}_k^i$);
13.     for each $P_x$ that has finished the assigned workload do
14.         Balance workload between $P_j$ and $P_x$, where $P_j$ is the heaviest worker;
15.     Upon receiving new REEs from workers, $S_c$ updates $\Sigma_i$ to $\Sigma_{i+1}$;
        update $\mathsf{T}_k^i$ to $\mathsf{T}_k^{i+1}$; broadcast $\mathsf{T}_k^{i+1}$ to all workers;   $i := i + 1$;
16. return $\Sigma_i$;
```

Figure 2: Algorithm PTopk-Miner

Workload balancing. At each superstep, if the workloads across workers are "skewed", *i.e.,* there is an idle worker P_x that has finished its assigned works, we re-distribute the workload to P_x from the heaviest worker P_j, in the following two steps.

(1) If there are more than one work unit in $\mathcal{W}_j$, P_j sends half of $\mathcal{W}_j$ (and the corresponding auxiliary structures) to P_x.
(2) If there is only one remaining work unit $w = \langle \mathcal{P}_{\mathsf{sel}}, \mathcal{P}_{\mathsf{re}}, p_0 \rangle$ in $\mathcal{W}_j$, we split this heavy unit into two smaller ones, namely $w' = \langle \mathcal{P}_{\mathsf{sel}} \cup \{p\}, \mathcal{P}_{\mathsf{re}} \setminus \{p\}, p_0 \rangle$ and $w'' = \langle \mathcal{P}_{\mathsf{sel}}, \mathcal{P}_{\mathsf{re}} \setminus \{p\}, p_0 \rangle$, where p is the predicate in $\mathcal{P}_{\mathsf{re}}$ with the highest processing order (see the processing order in Section 4.2), and send one of the two to P_x. Intuitively, it means that we divide the work unit w into two, *i.e.,* selecting p into $\mathcal{P}_{\mathsf{sel}}$ and excluding p from $\mathcal{P}_{\mathsf{sel}}$.

Parallel scalability. The parallel scalability is shown as follows.

Theorem 4: *Algorithm* PTopk-Miner *is parallelly scalable relative to the sequential algorithm* Topk-Miner. □

Proof. The time complexity of Topk-Miner is $t(|\mathcal{D}|, |\mathsf{RHS}|, \sigma, \delta) = O(\sum_{\varphi \in C(\mathcal{P}_0) \times \mathsf{RHS}} |\mathcal{D}|^{|\varphi|})$, where $C(\mathcal{P}_0)$ is the power set of $\mathcal{P}_0$ and $|\varphi|$ denotes the number of predicates in φ, since Topk-Miner examines the entire $C(\mathcal{P}_0)$ for each $p_0 \in$ RHS in the worst case.

We next show that the parallel runtime of PTopk-Miner is in $O(\frac{t(|\mathcal{D}|, |\mathsf{RHS}|, \sigma, \delta)}{n})$. In PTopk-Miner, S_c conducts workload assignment and heap maintenance, which maintains the global top-k by collecting REEs from each worker. The former takes $O(|\mathsf{RHS}|)$ time, while the latter takes $O(nk\log(k))$ time using merge-sort. Both are smaller than the discovery cost, which dominates the complexity.

The cost at each worker is dominated by the following: (a) transmit its top-k rules to the coordinator in time much less than $O(|\mathcal{D}|)$ since each rule is discovered from $\mathcal{D}$ and k is a small number; (b)

Name	Type	#tuples	#attributes	#relations
Adult [39, 47, 55]	real-life	32,561	15	1
Airport [47, 55]	real-life	55,113	18	1
Hospital [9, 13, 47, 55]	real-life	114,919	15	1
Inspection [47, 55, 61]	real-life	170,000	19	1
NCVoter [39, 47, 55]	real-life	1,681,617	12	1
DBLP [70]	real-life	1,799,559	18	3
Tax [9, 13, 18, 47, 55]	synthetic	10,000,000	15	1

Table 3: Dataset statistic

receive T_k from S_c in $\mathcal{O}(1)$ time; (c) balance its workload, where $\mathcal{O}(|\mathcal{D}|)$ data is sent to idle workers; and (d) locally perform discovery in $\mathcal{O}(\frac{t(|\mathcal{D}|,|\mathsf{RHS}|,\sigma,\delta)}{n})$ time, since the workload is evenly distributed in (c). Taken together, the parallel cost of PTopk-Miner is $\mathcal{O}(\frac{t(|\mathcal{D}|,|\mathsf{RHS}|,\sigma,\delta)}{n})$ in the worst-case. In practice, the pruning strategies in Section 4.2 effectively remove useless candidates. □

7 EXPERIMENTAL STUDY

Using real-life and synthetic data, we experimentally evaluated (1) the scalability of PTopk-Miner for top-k discovery and Anytime-Miner for anytime discovery, (2) the effectiveness of the proposed interestingness model, and (3) the effectiveness of top-k discovery.

Experimental setting. We start with the experimental setting.

Datasets. Following the setting in studies [47, 55], we used seven datasets as shown in Table 3. Adult, Airport, Hospital, Inspection and NCVoter are real-life datasets that are commonly used in the literature. We additionally used an academic dataset DBLP that has multiple relations, and a synthetic dataset Tax that is obtained by first duplicating tuples of the original tax data (1M) [9, 13] 10 times and then modifying their attributes using a program of [18].

ML models. We used two ML models as predicates in REEs. (1) For long textual attributes where logic predicates do not work well, we applied SentenceBert [60]. (2) For ER, we adopted ditto [45]. For the interestingness model $\mathcal{M}_{\text{interest}}$, we set the embedding size to 512. For UBSCORE, we adopt FFN with 3 **X**-dimensional hidden layers. We used Adam optimizer to train $\mathcal{M}_{\text{interest}}$ (resp. UBSCORE) with a batch-size of 20 (resp. **X**), and the learning rate is 0.001 (resp. **X**). We trained our model with 50 epochs on Tesla V100 GPU.

Baselines. We implemented the following, all in Java: (1) PTopk-Miner, (2) Anytime-Miner; (3) PTopk-Miner$_{\text{nop}}$, a variant of PTopk-Miner without using pruning strategy **[P3]**, which enumerates all minimal rules, computes their interestingness and returns the top-k ones; (4) PTopk-Miner$_{\text{dc}}$, another variant of PTopk-Miner, which discovers only DCs, *i.e.,* REEs on single relations without ML predicates; (5) DCFinder [55], a state-of-the-art DCs discovery algorithm, which mines all DCs that hold on the dataset; as shown in [55], it outperforms other DC discovery methods [55]. We parallelize it for a fair comparison and extend it to support constant predicates of the form $t.A = c$. We compared with PTopk-Miner$_{\text{nop}}$ to test the effectiveness of pruning strategies, and with PTopk-Miner$_{\text{dc}}$ and DCFinder for efficiency although they mine a special case of REEs.

We conducted experiments on a cluster of up to 21 virtual machines (one for the coordinator), each powered by 32GB RAM and 2 processors with 3.10 GHz. We ran the experiments 3 times, and report the average here. We do not include the time of loading datasets and constructing auxiliary structure, *i.e.,* PLI for all algorithms.

Experimental results. We next report our findings.

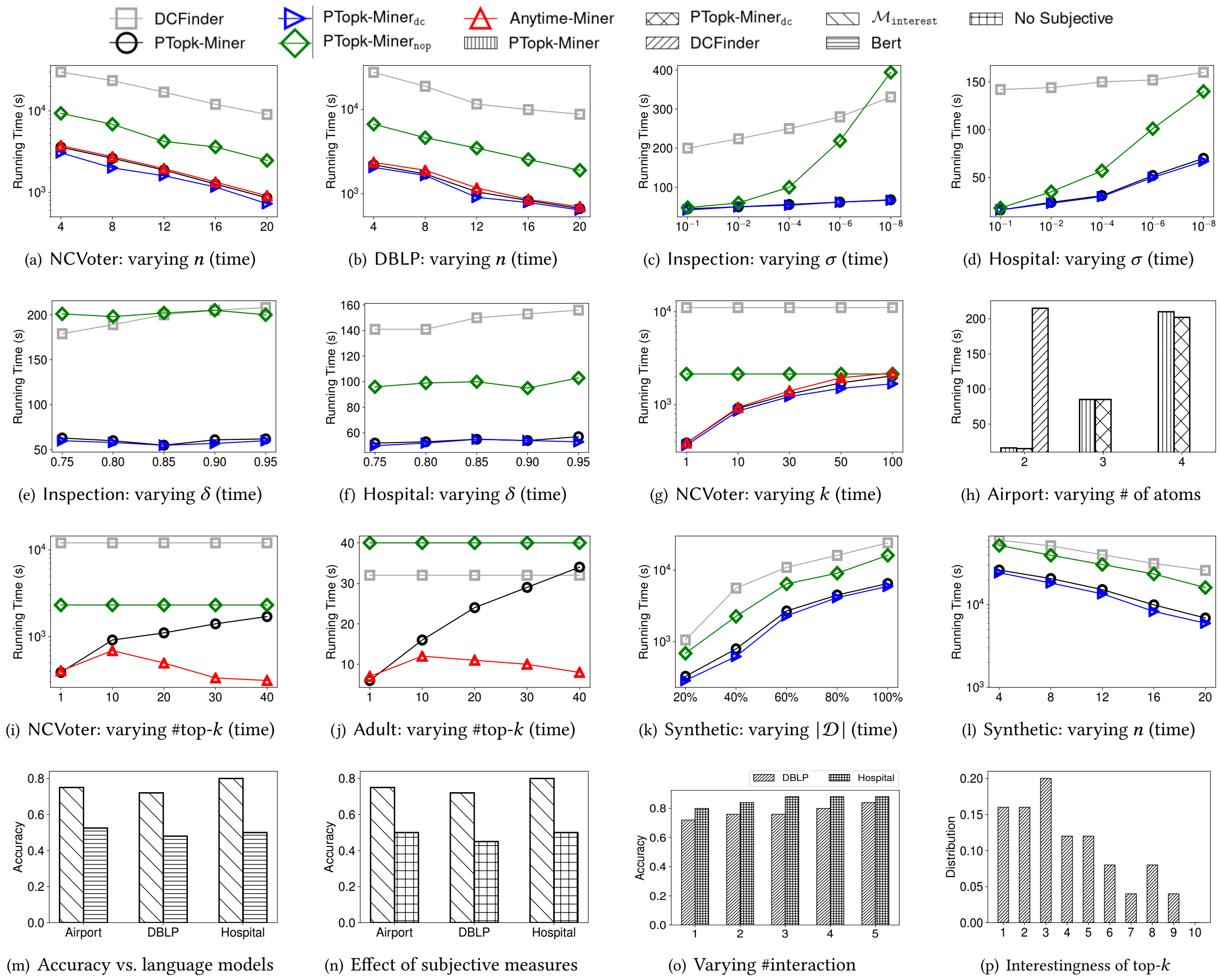
Exp-1: Scalability. We first evaluated the scalability of PTopk-Miner and Anytime-Miner vs. PTopk-Miner$_{\text{nop}}$, PTopk-Miner$_{\text{dc}}$ and DCFinder, by varying (1) the number n of machines, (2) the support threshold σ, (3) the confidence threshold δ, (4) the parameter k in top-k discovery, (5) the number of relation atoms in REEs, (6) the rounds of top-k discovery, and (7) the size of synthetic data. Unless stated explicitly, the default setting is $n = 20$, $\sigma = 10^{-6} \cdot |\mathcal{D}|^2$, $\delta = 0.75$, $k = 10$, and the default number of relation atoms in REEs is set to 2, for a fair comparison with PTopk-Miner$_{\text{dc}}$ and DCFinder that are restricted to bi-variable discovery. For large datasets DBLP, NCVoter and Tax, we included 5 predicates in RHS, which is typical in an application. We adopted the combined predicate processing order and used 4 objective measures and 1 subjective measures. For the lack of space, we only show the results on some of the datasets; the results on the other datasets are consistent.

Varying n. We varied the number n of machines from 4 to 20. As shown in Figures 3(a) and 3(b), (a) PTopk-Miner scales well with the increase of machines: it is 3.7 times faster when n varies from 4 to 20. (b) PTopk-Miner is feasible in practice. It takes 667s on DBLP when $n = 20$, as opposed to 8726s by DCFinder, although PTopk-Miner discovers REEs that carries ML predicates and relation atoms across multiple tables, which are not supported by DCFinder. (c) PTopk-Miner is 2.89 and 11.90 times faster than PTopk-Miner$_{\text{nop}}$ and DCFinder on average, respectively, up to 3.29 and 13.11 times. This verifies that our pruning strategies for finding top-k interesting rules effectively reduce the execution time. (d) Although PTopk-Miner$_{\text{dc}}$ is slightly faster than PTopk-Miner, it only discovers DCs, a special case of REEs. (e) Anytime-Miner is slightly slower than PTopk-Miner, since it has to maintain more rules in the heap and conduct implication checking to remove redundant rules (Section 5). Nonetheless, its advantage is evident when the users continuously require the next top-k results, as will be seen shortly.

Varying σ. Varying the support threshold σ from $10^{-1} \cdot |\mathcal{D}|^2$ to $10^{-8} \cdot |\mathcal{D}|^2$, we report the results in Figures 3(c) and 3(d). As expected, all algorithms take longer when σ is smaller since they need to examine more candidates, *e.g.,* PTopk-Miner$_{\text{nop}}$ is 8.2 times faster on Inspection when σ changes from $10^{-1} \cdot |\mathcal{D}|^2$ to $10^{-8} \cdot |\mathcal{D}|^2$. Nevertheless, PTopk-Miner is faster than PTopk-Miner$_{\text{nop}}$ and DCFinder under all values of σ, which is consistent with Figures 3(a) and 3(b). In particular, PTopk-Miner is less sensitive to σ, since it checks much less REEs than PTopk-Miner$_{\text{nop}}$ due to its pruning strategies. Anytime-Miner enjoys a similar trend as PTopk-Miner.

Varying δ. Varying the confidence threshold δ from 0.75 to 0.95, we report the results in Figures 3(e) and 3(f). As shown there, (a) almost all algorithms become faster given a smaller δ, *e.g.,* PTopk-Miner and Anytime-Miner are 1.10 times and 1.11 times faster, respectively, when δ varies from 0.95 to 0.75. This is because a higher δ indicates REEs with fewer violations, and this stronger constraint leads to more minimal REEs to be checked. (b) PTopk-Miner still consistently outperforms PTopk-Miner$_{\text{nop}}$ and DCFinder.

Varying k. Varying k from 1 to 100, Figure 3(g) shows that PTopk-Miner$_{\text{nop}}$ and DCFinder are indifferent to the values of k, since (a) PTopk-Miner$_{\text{nop}}$ mines all minimal REEs regardless of k, and (b) DCFinder discovers all minimal DCs subject to σ and δ. In contrast, PTopk-Miner takes much less time than the two due to its pruning strategies for top-k discovery; its cost increases slightly when k



(a) NCVoter: varying n (time)

(b) DBLP: varying n (time)

(c) Inspection: varying σ (time)

(d) Hospital: varying σ (time)

(e) Inspection: varying δ (time)

(f) Hospital: varying δ (time)

(g) NCVoter: varying k (time)

(h) Airport: varying # of atoms

(i) NCVoter: varying #top-k (time)

(j) Adult: varying #top-k (time)

(k) Synthetic: varying $|\mathcal{D}|$ (time)

(l) Synthetic: varying n (time)

(m) Accuracy vs. language models

(n) Effect of subjective measures

(o) Varying #interaction

(p) Interestingness of top-k

Figure 3: Performance evaluation

gets larger since more REEs have to be checked. Anytime-Miner is also faster than PTopk-Miner$_{\text{nop}}$ and DCFinder, by adopting a lazy evaluation strategy and skipping unnecessary REEs expansions.

Varying # of relation atoms. In Figure 3(h), we varied the number of relation atoms from 2 to 4. We observe that it takes longer to discover REEs with more relation atoms, as expected. Nonetheless, PTopk-Miner is still faster than DCFinder even when PTopk-Miner discovers more complicated REEs with 4 relation atoms while DCFinder only mines bi-variable DCs.

Varying #top-k. Fixing k=10, we varied the number #top-k of top-k results that users wish to see. Different from the lazy evaluation of Anytime-Miner, when users continue to find the next top-k, PTopk-Miner is executed with an increased value of k and an increased heap size so that it exactly returns the desired results. For instance, when #top-k = 4, PTopk-Miner discovers top-40 REEs.

Results on NCVoter and Adult are shown in Figures 3(i) and 3(j), respectively, when varying #top-k from 1 and 4. For the first top-k REEs, there is no big difference in the runtime between Anytime-Miner and PTopk-Miner. However, the advantage of Anytime-Miner over PTopk-Miner is more evident when the users want to see more top-k results, *e.g.,* when one asks for the 4th top-10 REEs, Anytime-Miner is 5.46 times and 4.25 times faster then PTopk-Miner on NCVoter and Adult, respectively. Intuitively, this is because Anytime-Miner maintains partial results and is more efficient to resume the discovery when needed. After Anytime-Miner accumulates sufficient partial results, its runtime may even decrease for larger #top-k. This shows the effectiveness of Anytime-Miner when the users continue to explore the next top-k rules.

Using large Tax synthetic data $\mathcal{D}$ (15 attributes and 10M tuples), we tested the impact of the size $|\mathcal{D}|$ and the number n of machines.

Varying $|\mathcal{D}|$ (synthetic). In Figure 3(k), we varied the scaling factor of $\mathcal{D}$ from 20% to 100%, *i.e.,* we changed the number of relations and tuples per relation from 2 million to 10 million. As shown there, all algorithms take longer, as expected. Consistent with the previous test results, PTopk-Miner still outperforms PTopk-Miner$_{\text{nop}}$ and DCFinder. It takes 1.8h when $\mathcal{D}$ has 10M tuples, as opposed to 4.4h and 6.5h by PTopk-Miner$_{\text{nop}}$ and DCFinder, respectively.

Varying n (synthetic). Fixing the data size $|\mathcal{D}|$ as 10M, we varied the number n of machines from 4 to 20. The results are reported in Figure 3(l) and are consistent with those in Figures 3(a) and 3(b). PTopk-Miner is 3.8 times faster when n varies from 4 to 20.

Exp-2: Effectiveness of the interestingness model. In this set of experiments, we studied the effectiveness of our interestingness model, by (1) comparing the accuracy of $\mathcal{M}_{\text{interest}}$ against the state-of-the-art language models, (2) evaluating the effectiveness of subjective measures, and (3) varying the rounds of interaction.

Accuracy vs. language models. We compared our interestingness model against Bert [14], which is implemented as the binary classification model with pairs of rules as input. We constructed a test set of rule pairs and asked the user to label the relative interestingness for each rule pair. Then, the model accuracy is measured by the percentage of rule pairs whose relative interestingness is correctly identified by the model. As reported in Fig. 3(m), our model consistently outperforms state-of-the-art language models in accuracy on all datasets, *e.g.,* on Hospital, our accuracy is 0.80, 30% higher than the language model. This is mainly due to unique features such as token segmentation and permutation invariant of logic rules, which are unable to be captured by existing language models.

Effectiveness of subjective measures. We next studied the effectiveness of subjective measures by comparing the accuracy of our interestingness model with and without subjective measures. As shown in Fig. 3(n), the results verify that objective measures alone cannot meet the users' need well since different users have diverse preference, *e.g.,* on DBLP, introducing subjective measures improves the accuracy by 27%, which clearly demonstrates the effectiveness of incorporating subjective measures in discovery.

Varying #interaction. To verify the usefulness of active learning, we report the accuracy by varying the number #r of rounds of interaction in Figure 3(o). With larger #r, the accuracy increases, *e.g.,* after 5 rounds, the accuracy changes from 0.72 to 0.84 on DBLP. This said, after 5 rounds, the accuracy gets stable since the model has accumulated enough training data. We find that it typically requires 5 rounds of interactions to achieve a stable accuracy.

Interestingness of top-k REEs. In this set of experiment, we mined top-10 REEs using PTopk-Miner and asked 5 users to label 5 of them which they think are interesting. As shown Figure 3(p), we plot the percentage of the users' "votes" for each rule. It shows that rules ranked higher by PTopk-Miner are indeed more favored by the users. More specifically, the rules ranked first to fifth by PTopk-Miner are labeled as interesting by 76% of the users on average, as opposed to 24% for the rules ranked sixth to tenth. The user ranking justifies the semantic of top-k interesting rule discovery.

Remark. We adopt ELMo to initialize token embeddings and finetune them to generate the rule embedding with the proposed model. It is effective in case of scarcity of rules since initial ELMo are pretrained on massive corpora. Our model contains only hundreds of parameters and is easily learned without much data.

Exp-3: Effectiveness of PTopk-Miner**.** We manually examined REEs discovered by PTopk-Miner from DBLP and Airport. Below are three REEs with support above 10 and confidence above 0.85.

(1) $\text{Author}_{\text{DBLP}}(t_0) \wedge \text{Author}_{\text{DBLP}}(t_1) \wedge \mathcal{M}_{\text{Bert}}(t_0.\text{Affiliate}, t_1.\text{Affiliate}) \wedge \mathcal{M}_{\text{ditto}}(t_0, t_1) \rightarrow t_0.\text{Name} = t_1.\text{Name}$. This REE states that if two authors have similar affiliations and they semantically match, then they have the same name. The rule employs ML models for similarity checking (Bert) and ER (ditto).

(2) $\text{Paper}_{\text{DBLP}}(t_0) \wedge \text{Paper}_{\text{DBLP}}(t_1) \wedge \mathcal{M}_{\text{Bert}}(t_0.\text{title}, t_1.\text{title}) \wedge t_0.\text{year} = t_1.\text{year} \wedge t_0.\text{venue} = t_1.\text{venue} \rightarrow \mathcal{M}_{\text{ditto}}(t_0.\bar{A}, t_1.\bar{A})$, where $\bar{A}$ is the set of all attributes in relation Paper of DBLP. This REE explores the possibility of interpreting an ML predication for ER by logic predicates and ML models. It says that two papers are predicted to match by an ML model because they have matching venue, year and title attributes. Note that the REE interprets the ML prediction of $\mathcal{M}_{\text{ditto}}$ with another ML predicate $\mathcal{M}_{\text{Bert}}$.

(3) $\text{Airport}(t_0) \wedge \text{Airport}(t_1) \wedge \text{Airport}(t_2) \wedge t_0.\text{iso\_region} = t_2.\text{iso\_region} \wedge t_1.\text{iata\_code} = t_2.\text{iata\_code} \wedge t_0.\text{municipality} = t_1.\text{municipality} \wedge t_0.\text{longitude\_deg} = t_2.\text{longitude\_deg} \rightarrow t_0.\text{iso\_region} = t_1.\text{iso\_region}$. This REE collectively checks consistency by using three relation atoms. Current DC discovery algorithms do not support rules with more than two relation atoms.

(4) $\text{Paper}_{\text{DBLP}}(t_0) \wedge \text{Paper}_{\text{DBLP}}(t_1) \wedge t_0.\text{year} \neq t_1.\text{year} \rightarrow t_0.\text{id} \neq t_1.\text{id}$. It distinguishes papers that are published in different years.

Summary. We find the following. (1) Top-k REEs discovery speeds up PTopk-Miner$_{\text{dc}}$ and DCFinder by 2.6 and 8.4 times on average, respective, up to 4.2 and 13.1 times. When $n = 20$, it takes less than 670s to mine top-10 REEs from DBLP that has 3 relations, 18 attributes and 1.8M tuples, as opposed to 3043s and 8726s by PTopk-Miner$_{\text{nop}}$ and DCFinder, respectively. (2) PTopk-Miner scales well with various parameters; *e.g.,* when the support threshold σ decreases on Inspection, PTopk-Miner and Anytime-Miner take only slightly longer. (3) PTopk-Miner is parallelly scalable: on average, it is 3.6 times faster when the number n of machines varies from 4 to 20. (4) The lazy evaluation strategy of Anytime-Miner is effective when the users continuously want the next top-k results: Anytime-Miner is 4.85 times faster than PTopk-Miner, when the users want the 4th top-10 REEs. (5) Our pruning strategies are effective, *e.g.,* reducing the the runtime of PTopk-Miner by 2.89 times on the Tax data of 10M tuples. (6) Our interestingness model is on average 25.5% more accurate than the state-of-the-art language models. In particular, the subjective measures introduced in the model improves the accuracy from 0.48 to 0.76. (7) PTopk-Miner is capable of finding truly interesting REEs from real-life data.

8 CONCLUSION

The novelty of the work consists of the following: (1) an interestingness model with both objective and subjective measures; (2) an active-learning method to learn the subjective model and weight vector; (3) a top-k algorithm for discovering REEs, which subsume CFDs, DCs and MDs as special cases; (4) an anytime algorithm to continuously mine the next top-k rules via lazy evaluation, and (5) parallelization of the algorithms with the parallel scalability. Our experimental study has shown that the method is promising.

One topic for future work is to study incremental top-k rule discovery in response to updates to both users interest and datasets. Another topic is to integrate top-k algorithm and data sampling, to further speed up the discovery process, with accuracy guarantees.

REFERENCES

[1] [n.d.]. Full version. *http://homepages.inf.ed.ac.uk/wenfei/papers/GFix.pdf.*

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

[5] Zeinab Bahmani and Leopoldo E. Bertossi. 2017. Enforcing Relational Matching Dependencies with Datalog for Entity Resolution. In *FLAIRS*.

[6] Laure Berti-Équille, Hazar Harmouch, Felix Naumann, Noël Novelli, and Saravanan Thirumuruganathan. 2018. Discovery of Genuine Functional Dependencies from Relational Data with Missing Values. *PVLDB* 11, 8 (2018), 880–892.

[7] Leopoldo E. Bertossi, Solmaz Kolahi, and Laks V. S. Lakshmanan. 2013. Data Cleaning and Query Answering with Matching Dependencies and Matching Functions. *Theory Comput. Syst.* (2013).

[8] Indrajit Bhattacharya and Lise Getoor. 2007. Collective entity resolution in relational data. *TKDD* (2007).

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

[10] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).

[11] Fei Chiang and Renee Miller. 2008. Discovering Data Quality Rules. In *VLDB*.

[12] Vassilis Christophides, Vasilis Efthymiou, Themis Palpanas, George Papadakis, and Kostas Stefanidis. 2021. An Overview of End-to-End Entity Resolution for Big Data. *ACM Comput. Surv.* 53, 6 (2021), 127:1–127:42.

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

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*. 4171–4186.

[15] Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, and Denny Zhou. 2019. Neural Logic Machines. In *ICLR*.

[16] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq R. Joty, Mourad Ouzzani, and Nan Tang. 2018. Distributed Representations of Tuples for Entity Resolution. *PVLDB* (2018).

[17] Wenfei Fan, Hong Gao, Xibei Jia, Jianzhong Li, and Shuai Ma. 2011. Dynamic constraints for record matching. *VLDB J.* 20, 4 (2011), 495–520.

[18] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2008. Conditional functional dependencies for capturing data inconsistencies. *TODS* 33, 2 (2008), 6:1–6:48.

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

[20] Wenfei Fan, Floris Geerts, Nan Tang, and Wenyuan Yu. 2014. Conflict resolution with data currency and consistency. *J. Data and Information Quality* 5, 1-2 (2014), 6:1–6:37.

[21] Wenfei Fan, Ping Lu, and Chao Tian. 2020. Unifying Logic Rules and Machine Learning for Entity Enhancing. *Science China Information Sciences* (2020).

[22] Wenfei Fan, Chao Tian, Yanghao Wang, and Qiang Yin. 2021. Parallel Discrepancy Detection and Incremental Detection. *PVLDB* (2021).

[23] Wenfei Fan, Xin Wang, Yinghui Wu, and Jingbo Xu. 2015. Association Rules with Graph Patterns. *PVLDB* 8, 12 (2015), 1502–1513.

[24] Peter A Flach and Iztok Savnik. 1999. Database dependency discovery: A machine learning approach. *AI communications* 12, 3 (1999), 139–160.

[25] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. 2007. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics* 9, 3 (2007), 432–441.

[26] Cheng Fu, Xianpei Han, Le Sun, Bo Chen, Wei Zhang, Suhui Wu, and Hao Kong. 2019. End-to-end multi-perspective matching for entity resolution. In *IJCAI*. 4961–4967.

[27] Eve Garnaud, Nicolas Hanusse, Sofian Maabout, and Noël Novelli. 2014. Parallel mining of dependencies. In *HPCS*. IEEE, 491–498.

[28] Gartner. 2018. How to create a business case for data quality improvement. *https://www.gartner.com/smarterwithgartner/how-to-create-a-business-case-for-data-quality-improvement/.*

[29] Chang Ge, Ihab F. Ilyas, and Florian Kerschbaum. 2019. Secure Multi-Party Functional Dependency Discovery. *PVLDB* 13, 2 (2019), 184–196.

[30] Liqiang Geng and Howard J. Hamilton. 2006. Interestingness measures for data mining: A survey. *ACM Comput. Surv.* 38, 3 (2006), 9.

[31] Lise Getoor and Ashwin Machanavajjhala. 2012. Entity resolution: tutorial. *VLDB* (2012).

[32] Lukasz Golab, Howard Karloff, Flip Korn, Divesh Srivastava, and Bei Yu. 2008. On generating near-optimal tableaux for conditional functional dependencies. *VLDB* (2008).

[33] Hado van Hasselt, Arthur Guez, and David Silver. 2016. Deep Reinforcement Learning with Double Q-Learning. In *AAAI* (Phoenix, Arizona) *(AAAI'16)*. 2094–2100.

[34] Alireza Heidari, Joshua McGrath, Ihab F. Ilyas, and Theodoros Rekatsinas. 2019. HoloDetect: Few-Shot Learning for Error Detection. In *SIGMOD*.

[35] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. 1999. TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies. *Comput. J.* (1999).

[36] 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.

[37] loannis Koumarelas, Thorsten Papenbrock, and Felix Naumann. 2020. MDedup: Duplicate detection with matching dependencies. *PVLDB* 13, 5 (2020), 712–725.

[38] Sebastian Kruse and Felix Naumann. 2018. Efficient Discovery of Approximate Dependencies. *PVLDB* 11, 7 (2018), 759–772.

[39] Sebastian Kruse and Felix Naumann. 2018. Efficient discovery of approximate dependencies. *VLDB* (2018).

[40] Clyde P Kruskal, Larry Rudolph, and Marc Snir. 1990. A complexity theory of efficient parallel algorithms. *TCS* (1990).

[41] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942* (2019).

[42] Bing Li, Wei Wang, Yifang Sun, Linhan Zhang, Muhammad Asif Ali, and Yi Wang. 2020. GraphER: Token-Centric Entity Resolution with Graph Convolutional Neural Networks.. In *AAAI*. 8172–8179.

[43] Weibang Li, Zhanhuai Li, Qun Chen, Tao Jiang, and Hailong Liu. 2015. Discovering functional dependencies in vertically distributed big data. In *WISE*. 199–207.

[44] Weibang Li, Zhanhuai Li, Qun Chen, Tao Jiang, and Zhilei Yin. 2016. Discovering approximate functional dependencies from distributed big data. In *APWeb*. 289–301.

[45] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep Entity Matching with Pre-Trained Language Models. *arXiv preprint arXiv:2004.00584* (2020).

[46] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[47] Ester Livshits, Alireza Heidari, Ihab F. Ilyas, and Benny Kimelfeld. 2020. Approximate Denial Constraints. *PVLDB* 13, 10 (2020), 1682–1695.

[48] Stéphane Lopes, Jean-Marc Petit, and Lotfi Lakhal. 2000. Efficient discovery of functional dependencies and Armstrong relations. In *EDBT*. Springer, 350–364.

[49] Mohammad Mahdavi, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2019. Raha: A Configuration-Free Error Detection System. In *SIGMOD*. 865–882.

[50] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nat.* (2015).

[51] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep learning for entity matching: A design space exploration. In *SIGMOD*.

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

[53] Thorsten Papenbrock, Jens Ehrlich, Jannik Marten, Tommy Neubert, Jan-Peer Rudolph, Martin Schönberg, Jakob Zwiener, and Felix Naumann. 2015. Functional dependency discovery: An experimental evaluation of seven algorithms. *PVLDB* 8, 10 (2015), 1082–1093.

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

[55] 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.

[56] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proc. of NAACL*.

[57] Meng Qu, Junkun Chen, Louis-Pascal A. C. Xhonneux, Yoshua Bengio, and Jian Tang. 2020. RNNLogic: Learning Logic Rules for Reasoning on Knowledge Graphs. *CoRR* abs/2010.04029 (2020).

[58] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[59] Thomas C. Redman. 2016. Bad Data Costs the U.S. $3 Trillion Per Year. Harvard Business Review. *https://hbr.org/2016/09/bad-data-costs-the-u-s-3-trillion-per-year.*

[60] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *EMNLP-IJCNLP*. 3980–3990.

[61] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *PVLDB* (2017).

[62] Theodoros Rekatsinas, Manas Joglekar, Hector Garcia-Molina, Aditya G. Parameswaran, and Christopher Ré. 2017. SLiMFast: Guaranteed Results for Data Fusion and Source Reliability. In *SIGMOD*. 1399–1414.

[63] Hemant Saxena, Lukasz Golab, and Ihab F Ilyas. 2019. Distributed discovery of functional dependencies. In *ICDE*. IEEE, 1590–1593.

[64] Hemant Saxena, Lukasz Golab, and Ihab F Ilyas. 2019. Distributed implementations of dependency discovery algorithms. *PVLDB* 12, 11 (2019), 1624–1636.

[65] Philipp Schirmer, Thorsten Papenbrock, Ioannis Koumarelas, and Felix Naumann. 2020. Efficient Discovery of Matching Dependencies. *RODS* 45, 3 (2020), 1–33.

[66] 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*.

[67] Shaoyun Shi, Hanxiong Chen, Weizhi Ma, Jiaxin Mao, Min Zhang, and Yongfeng Zhang. 2020. Neural Logic Reasoning. In *CIKM*. 1365–1374.

[68] 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.

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

[70] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. ArnetMiner: Extraction and Mining of Academic Social Networks. In *KDD'08*. 990–998.

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

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

[73] Renzhi Wu, Sanya Chaba, Saurabh Sawlani, Xu Chu, and Saravanan Thirumuruganathan. 2020. ZeroER: Entity Resolution using Zero Labeled Examples. In *SIGMOD*. 1149–1164.

[74] 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*.

[75] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *NeurIPS*. 5753–5763.

[76] H Yao, H Hamilton, and C Butz. 2002. Fd_mine: discovering functional dependencies in a database using equivalences, Canada. In *IEEE ICDM*. 1–15.

[77] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. 2017. Deep Sets. In *Advances in Neural Information Processing Systems*. 3391–3401.

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

[79] Chen Zhao and Yeye He. 2019. Auto-EM: End-to-end Fuzzy Entity-Matching using Pre-trained Deep Models and Transfer Learning. In *WWW*. 2413–2424.

Algorithm ActiveLearner

Input: A rule pool $\mathcal{S}_{\text{REEs}}$, accuracy acc_{min}, maximum # of iterations iter, # of initial rule pairs M, and # of rule pairs N to label in each iteration.

Output: The interestingness model $\mathcal{M}_{\text{interest}}$.

```
1. Construct $\mathcal{C}_{\text{REEs}}$, consisting of $M$ pairs of rules sampled from $\mathcal{S}_{\text{REEs}}$;
2. Ask the user to label $\mathcal{D}_{\text{REEs}}$;
3. Split $\mathcal{C}_{\text{REEs}}$ into $\mathcal{C}_{\text{train}}$ and $\mathcal{C}_{\text{valid}}$; step := 0; Initialize ML model $\mathcal{M}_{\text{interest}}$;
4. while step $\leq$ iter do
5.     Incrementally train $\mathcal{M}_{\text{interest}}$ using the current $\mathcal{C}_{\text{train}}$;
6.     $\text{acc}_{\text{valid}}$ := Evaluate($\mathcal{M}_{\text{interest}}$, $\mathcal{C}_{\text{valid}}$);
7.     if $\text{acc}_{\text{valid}} > \text{acc}_{\text{min}}$ then
8.         break ;
9.     for each $\varphi \in \mathcal{S}_{\text{REEs}}$ do
10.        Compute the interestingness of $\varphi$, i.e., $\mathcal{M}_{\text{interest}}(\varphi)$;
11.    Select a set $\Delta\mathcal{C}_1$ of top-$\lceil \frac{N}{2} \rceil$ rule pairs $\langle \varphi_i, \varphi_j \rangle$ in $\mathcal{S}_{\text{REEs}}$ with the
       smallest differences of interestingness $|\mathcal{M}_{\text{interest}}(\varphi_i) - \mathcal{M}_{\text{interest}}(\varphi_j)|$;
12.    Randomly select a set $\Delta\mathcal{C}_2$ of $\lfloor \frac{N}{2} \rfloor$ rule pairs from $\mathcal{S}_{\text{REEs}}$;
13.    $\Delta\mathcal{C} := \Delta\mathcal{C}_1 \cup \Delta\mathcal{C}_2$ and ask the user to label $\Delta\mathcal{C}$;
14.    $\mathcal{C}_{\text{train}} := \mathcal{C}_{\text{train}} \cup \Delta\mathcal{C}$; step := step + 1;
15. return $\mathcal{M}_{\text{interest}}$;
```

Figure 4: Algorithm ActiveLeaner

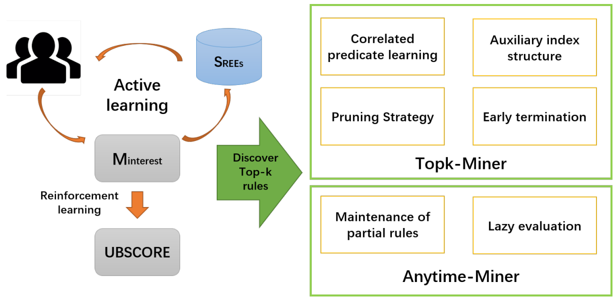


Figure 5: Rule Discovery Architecture

APPENDIX A: MORE OBJECTIVE MEASURES

Objective measures. We list a few objective measures used in the interestingness model as follows.

Minimality. An REE $\varphi : X \to p_0$ over $\mathcal{R}$ is said to be *trivial* if $p_0 \in X$. In the rest of this paper, we only consider non-trivial REEs.

An REE $\varphi : X \to p_0$ is *left-reduced* on $\mathcal{D}$ if φ is σ-frequent, δ-confident and moreover, there exists no REE φ' such that $\varphi' \preceq \varphi$ and φ' is σ-frequent and δ-confident. Intuitively, it means no predicate in X can be removed, *i.e.,* the minimality of predicates.

A *minimal* REE φ on $\mathcal{D}$ is a non-trivial and left-reduced REE. Intuitively, there is no redundancy in a minimal REE.

Attribute diversity. To deduce a consequence p_0, we want different preconditions X to include diverse attributes so that if noises appear in some attributes, we can still use other attributes to deduce p_0. To do this, we maintain a counter $\text{ct}_{p_0}(A)$ for each A in $\mathcal{R}$ and increase it by 1 whenever an REE whose precondition involves A is discovered. Then, the *diversity* of an REE $\varphi : X \to p_0$ is $\text{div}(\varphi) = 1/\max\{\text{ct}_{p_0}(A) \mid A \text{ is an attribute that appears in } X\}$.

Succinctness. We define the *succinctness* of REEs as $\text{suc}(\varphi) = 1/|X|$ where $|X|$ denotes the number of predicates used in X. It is widely agreed that REEs with larger succinctness are easier to understand and are statistically more likely to be true [65].

APPENDIX B: ACTIVE LEARNING

As shown in Figure 4, the active learner starts by randomly sampling a few pairs of rules from $\mathcal{S}_{\text{REEs}}$; it lets the user label them to generate the initial training set $\mathcal{C}_{\text{train}}$ of rules and the initial validation set $\mathcal{C}_{\text{valid}}$, where $\mathcal{C}_{\text{train}}$ is used to train $\mathcal{M}_{\text{interest}}$ and $\mathcal{C}_{\text{valid}}$ is used to evaluate its accuracy (lines 1-3).

We then iteratively learn $\mathcal{M}_{\text{interest}}$ (lines 4-15). More specifically, we first train $\mathcal{M}_{\text{interest}}$ using the current training data $\mathcal{C}_{\text{train}}$. Then we actively select more rule pairs from $\mathcal{S}_{\text{REEs}}$ for users to label, aiming to improve the accuracy. To do this, we use the current $\mathcal{M}_{\text{interest}}$ to compute the interestingness score for each rule in $\mathcal{S}_{\text{REEs}}$. Then we select rule pairs in $\mathcal{S}_{\text{REEs}}$ that $\mathcal{M}_{\text{interest}}$ cannot distinguish well, *i.e.,* pairs that have the smallest differences in interestingness scores, and send them to users for labeling. Intuitively, asking the user to label such pairs is more beneficial than labeling pairs with a clear margin, since it provides more information. To increase the diversity, we also randomly select a few rule pairs from $\mathcal{S}_{\text{REEs}}$ (line 12) for users to label. Finally, we include the newly labeled data in $\mathcal{C}_{\text{train}}$ (note that none of the newly labeled rules can be in $\mathcal{C}_{\text{valid}}$).

The process proceeds until it reaches the maximum number of iterations or the accuracy of $\mathcal{M}_{\text{interest}}$, evaluated by using $\mathcal{C}_{\text{valid}}$, reaches the minimum accuracy specified by users. We find it often suffices for users to label 50 rule pairs in 5 rounds of interaction.

Example 11: Consider a rule pool $\mathcal{S}_{\text{REEs}}$ of size 100. Assume $M = 20$ and $N = 10$, we first sample 20 pairs of rules from $\mathcal{S}_{\text{REEs}}$ and ask users to give labels, using which to initialize ML model $\mathcal{M}_{\text{interest}}$. Next, ask users to label 10 pairs of rules including some with the smallest differences in interestingness score and some that are random choosed, retrain $\mathcal{M}_{\text{interest}}$, and calculate the accuracy at each iteration. The process iterates until iter or acc_{min} are reached. □

APPENDIX C: OVERVIEW OF RULE DISCOVERY

Figure 5 shows the architecture of top-k rule discovery. First, we train the rule interestingness model $\mathcal{M}_{\text{interest}}$ using active learning. Then, another ML model UBSCORE is learned by interacting with $\mathcal{M}_{\text{interest}}$ in a reinforcement learning manner, such that it could be used as an estimator of the interestingness upper bound in the rule discovery procedure. Finally two components, Topk-Miner and Anytime-Miner, are executed to discover the top-k interesting rules.