Discovering Top-k Relevant and Diversified Rules

WENFEI FAN, Beihang University, China, Shenzhen Institute of Computing Sciences, China, and University of Edinburgh, United Kingdom
ZIYAN HAN, Beihang University, China
MIN XIE, Shenzhen Institute of Computing Sciences, China
GUANGYI ZHANG*, Shenzhen Technology University, China

This paper studies the problem of discovering top-k relevant and diversified rules. Given a real-life dataset, it is to mine a set of k rules that are as close to users' interest as possible, and meanwhile, as diverse to each other as possible. It aims to reduce excessive irrelevant rules commonly returned by rule discovery. As a testbed, we consider Entity Enhancing Rules (REEs), which subsume popular data quality rules as special cases. We train a relevance model to learn users' prior knowledge, rank rules based on users' need, and propose four diversity measures to assess the diversity between rules. Based on these measures, we formulate a new discovery problem. We show that the bi-criteria discovery problem is NP-complete and hard to approximate. This said, we develop a practical algorithm for the problem, and prove its approximation bounds under certain conditions. Moreover, we develop optimization techniques to speed up the process, and parallelize the algorithm such that it guarantees to reduce runtime when given more processors. Using real-life data, we empirically verify that on average, the top-10 REEs discovered by our algorithm is able to catch 77.5% of errors detected by the entire set Σ_{all} of REEs and achieve F1 = 0.74 for real error detection; moreover, discovering top-ranked REEs is 62.4X faster than mining Σ_{all}.

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

Additional Key Words and Phrases: Rule discovery, top-k, diversified

ACM Reference Format:
Wenfei Fan, Ziyan Han, Min Xie, and Guangyi Zhang. 2024. Discovering Top-k Relevant and Diversified Rules. *Proc. ACM Manag. Data* 2, 4 (SIGMOD), Article 195 (September 2024), 28 pages. https://doi.org/10.1145/3677131

1 Introduction

Logic rules have found prevalent use in data cleaning, association analysis, knowledge discovery, online recommendation, drug discovery and manufacturing industry, among other things [31]. To make practical use of rules, we have to discover high-quality rules from real-life data. A number of rule discovery methods have been studied [8, 11, 12, 17, 23, 26, 34–36, 43, 49, 51, 57, 70, 71, 78, 81, 97–102, 106, 107, 109, 112, 115, 117], which typically take a dataset $\mathcal{D}$ as input, and mine or learn rules from $\mathcal{D}$ such that the rules have support and confidence above predefined thresholds; here support measures how often a rule can be applied, and confidence assesses how strongly the precondition and consequence are associated.

*Corresponding author

Authors' Contact Information: Wenfei Fan, Beihang University, China and Shenzhen Institute of Computing Sciences, China and University of Edinburgh, United Kingdom, wenfei@inf.ed.ac.uk; Ziyan Han, Beihang University, China, hanzy@act.buaa.edu.cn; Min Xie, Shenzhen Institute of Computing Sciences, China, xiemin@sics.ac.cn; Guangyi Zhang, Shenzhen Technology University, China, zhangguangyi@sztu.edu.cn.

ACM 2836-6573/2024/9-ART195
https://doi.org/10.1145/3677131

Rock [7, 14] is an industry system that combines rules and ML for data cleaning and association analyses. From working with its developers and users, we find that practitioners in industry raise the following issues about the previous methods for rule discovery.

(1) Excessive irrelevant rules. The prior methods typically return the entire set of rules from $\mathcal{D}$ with high enough support and confidence. The set is often quite large. For instance, from a small dataset with 27 attributes and 368 tuples, 128,726 functional dependencies (FDs) are found [80]. The users are often overwhelmed by the excessive rules and have to spend a huge amount of time to manually inspect and select rules that fit their need. Moreover, practitioners often have developed prior knowledge about their applications, and even have accumulated dozens of rules after years of practice. Thus they only want rules that fit their need and are unknown/novel to them, not those that are already known or "trivial"/"common-sense", *e.g.,* only 10% rules discovered are considered as novel in bank reporting [95].

(2) Extensive cost. It is prohibitively expensive to mine the entire set of rules from $\mathcal{D}$. The prior methods enumerate candidate rules, for each rule, compute its support and confidence by validating it on the entire dataset, *e.g.,* it takes >3 hours on a dataset with 1.68M tuples even with 20 machines (Section 6). Moreover, it is often costly to apply the large number of mined rules to datasets, *e.g.,* [92].

(3) Much of a muchness. To reduce irrelevant rules and discovery cost, top-k discovery has been studied, to find top-ranked rules based on objective measures (support and confidence) and subjective measures (relevance to users' need), from relations [36, 107] and graphs [40]. Unfortunately, they still fall short in industry, because the rules returned are often too "homogeneous" to each other and hence, are often considered redundant and non-interesting. As an evidence, a rule-based method for field search was adopted in a data platform, aiming to improve the diversity of the results [93].

Example 1: As reported in [94], a leading bank purchased data from external sources and found a large number of duplicated companies, leading to heavy verification effort, especially for companies established before 2015. This is because before 2015, there was no unified coding system, and companies used registration codes, organization codes and/or tax identifiers to identify themselves, which were prone to inconsistencies and inaccuracies.

The bank used the following *real rule* for entity resolution (ER).

φ_k : Two companies t and s refer to the same company if they have (a) *similar* names and (b) the same industry type (*e.g.,* retail).

This rule was useful since companies with similar names can refer to different entities (*e.g.,* Apple Inc. [3] and Apple Corps Ltd. [2]) and industry types give useful side information to make decisions.

Consider two real rules φ_1 and φ_1' newly discovered for ER, where φ_1 identifies two companies if they have same Unified Social Credit Code (USCC, the unique identifier issued for business in the country after 2015 [6]), and φ_1' does so if they have the *same* name and same industry type. One can see that φ_1' and φ_k are "homogeneous", differing only in how they compare company names. If two companies can be identified by φ_1', they can be trivially identified by φ_k. Therefore, φ_1' is "unsurprising" and "redundant" to the bank.

In contrast, φ_1 is more helpful than φ_1', since, *e.g.,* it uses a different set of attributes to identify companies and thus in principle, φ_1 can identify some companies that are *unknown* by using φ_k or φ_1'.

Based on support and confidence, φ_1' may also be returned by top-k algorithms [107] for its higher counts. With this comes the need for both relevance and diversity measures in rule discovery. □

As suggested by the example, the practitioners often want only rules that are (a) relevant to their need and are unknown or surprising to them, and (b) dissimilar to each other, and diverse enough to cover different categories or characterize different features.

These concerns raise several questions. How should we measure the relevance and diversity? Can we formulate a bi-criteria problem for discovering a set of rules that are on the one hand, relevant to users' need, and on the other hand, are dissimilar to each other? What is the complexity of this problem? Is it possible to develop an algorithm that is both accurate and scalable to large datasets?

Contributions & organization. This paper tackles these questions. For rules, we consider Entity Enhancing Rules (REEs) [37, 38], a class of rules being used by Rock. REEs subsume popular data quality rules as special cases, such as conditional functional dependencies (CFDs) [33], denial constraints (DCs) [13] and matching dependencies (MDs) [32]; moreover, REEs are collectively defined across multiple tables, and may embed ML models as predicates. After reviewing REEs in Section 2, we develop the following.

(1) A bi-criteria problem (Section 3). We advocate a different rule discovery paradigm. Given a set $\mathcal{D}$, we find k top-ranked REEs from $\mathcal{D}$ such that the rules are both relevant to users' need and diverse to each other. This reduces redundant rules and discovery cost.

We train a relevance model to learn users' need, and introduce four measures to assess the diversity of REEs. Based on the model and measures, we define a bi-criteria objective function and formulate the discovery problem for top-k relevant and diversified REEs. We show that the problem is NP-complete and hard to approximate.

(2) An approximate algorithm (Section 4). Despite the hardness, we develop an algorithm for the new discovery problem, denoted by TopKDivMiner. TopKDivMiner greedily mines relevant and diverse REEs in rounds. We show that for certain relevance and diversity measures, it ensures an $(1 - 1/e)$-approximation bound. Moreover, under certain conditions, it guarantees 4-approximation.

(3) Optimization and parallelization (Section 5). To make the algorithm practical, we develop strategies to speed up the discovery process. We propose pruning strategies within each greedy round and between rounds of TopKDivMiner. To scale with large datasets $\mathcal{D}$, we parallelize TopKDivMiner, denoted by PTopKDivMiner. We show that PTopKDivMiner is parallelly scalable [62], *i.e.,* they guarantee to reduce parallel runtime when more processors are used.

(4) Experimental study (Section 6). Using real-life datasets, we find: (a) Discovery of top-k relevant and diversified rules is efficient. PTopKDivMiner takes 881s to mine top-10 rules from 1.68M tuples; for all $k \leq 40$, it is on average 62.4X faster than mining the entire set Σ_{all}. (b) PTopKDivMiner is parallelly scalable: on average, it is 3.05X faster when 20 machines are used instead of 4. (c) The rules returned by PTopKDivMiner are relevant and diverse: top-10 rules are able to detect real "unknown-to-user" errors in datasets that were already cleaned by users' own methods, with F1 above 0.74, 39.62% higher than the top-k method of [36]; moreover, they catch 77.5% of errors detected by Σ_{all}, *i.e.,* top-10 rules are able to detect diverse errors.

Related work. We categorize the related work as follows.

Rule mining. Rule learning has been extensively studied, by greedily growing/pruning a predicate set [27, 47], logic programming [76], decision trees [19, 86–88], custom branch-and-bound algorithms [12, 56, 68], Bayesian methods and sampling [23, 97, 106], etc.

To learn multiple-variable rules such as functional dependencies (FDs) [9], DCs [13], CFDs [33], MDs [32], REEs [37], and their extensions [49, 51, 99, 101], it requires more sophisticated techniques. Their discovery typically follows levelwise search [57, 71, 78, 112], depth-first search (DFS) [8, 109] or a hybrid one [60, 81, 98]. The levelwise search, a form of breath-first search (BFS) initiated by Apriori [11] and its alternatives [74, 102, 115], has been modified to mine FDs and CFDs in, *e.g.,* TANE [57] and CTANE [34], respectively. Alternatively, DFS is more space-efficient, and has been

employed to mine, *e.g.,* CFDs by FastCFDs [34], DCs by FastDC [26] and DCFinder [83]. Sampling is essential for many of these methods to scale [17, 35, 70]. In addition, there have been ML-based attempts to learn rules [24, 43, 73, 117], or to replace certain components of rule discovery [25, 37]. Our algorithms adopt the BFS approach.

Top-k diversified algorithms. The problem of diversity maximization was first studied as subset selection by [89], and a variety of diversity notions were proposed in [22]. Other than diversity, a second modular objective of the selected subset is simultaneously optimized [52]. More general objectives also appeared [18, 28, 84]. Other statistical definitions of diversity, *e.g.,* the coefficient of variation [15] and balancing counts among disjoint classes [54], also exist.

Incorporating diversity into top-k query processing is of great interest for reducing redundancy, such as finding diverse top-k frequent patterns [110], matched subgraphs [39, 111] and cliques [114]. Diversity is particularly helpful to information retrieval [21], where user clicks can be used as signals to automatically learn a diversity model [58, 69]. There has also been work that diversifies top-k results via post-processing [85]. Theory-wise, a standard setup has been analyzed where the top-k results are aggregated among multiple given sorted lists [45]. The complexity of diversification and query evaluation has also been studied [29, 52, 105].

Closer to this work is top-k rule discovery [36, 96, 103, 107, 108]. [107] proposes a general scheme for top-k association rule discovery, which is amenable to more pruning operations than searching over a fixed-structure. [96, 108] look for top-k predicates that each cover outliers in the data whose removal causes dramatic change to the aggregation score. [103] mines special top-k preference rules from user rated reviews. [36] finds top-k rules with an ML-based relevance model that learns user preference by interactive pairwise comparisons. [83] prunes and ranks rules by three interesting measures: succinctness, coverage and degree of approximation; more interestingness measures are discussed in [50, 54]. On the other hand, early research on diverse rules focuses on a limited form of diversity [44], or coverage-based redundancy [63, 116]. [40] generalizes conventional association rules on itemsets to graph-pattern association rules (GPARs); it also studies diversified top-k GPARs, but the diversity is specialized for non-overlap between graph entities.

Among these, [36] is the closest one, which only focuses on relevance but does not study diversity. In contrast, we incorporate both relevance and diversity into rule ranking, to avoid excessive rules, extensive cost and homogeneous results; this introduces novel challenges in rule discovery. To the best of our knowledge, this work makes the first effort to study discovery of top-k relevant and diversified rules. (1) We propose various diversity measures from syntactic, structural and semantic perspectives, that are more general than the prior ones [40, 44, 116]. (2) We train a relevance model that is more accurate and easier to use, differing from traditional interesting measures [40, 107], interactive selection by users [10] and recent ML models [30, 36]. (3) We formulate the bi-criterion discovery problem, following the common form for a multi-objective optimization problem [10, 63]. We settle its complexity and approximation hardness. (4) We develop the first algorithms for the new discovery problem, with approximation bounds under certain conditions, pruning strategies, and parallel scalability to ensure accuracy and efficiency.

2 Collective Rules with ML Models

We next present entity enhancing rules (REEs) defined in [37, 38].

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

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

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

where $\oplus$ is an operator in $\{=, \neq, <, \leq, >, \geq\}$. Following tuple relational calculus [9], (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) $t.A$ is 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; (d) $t.A \oplus s.B$ is *a variable predicate* that compares *compatible* attributes $t.A$ and $s.B$, *i.e.,* t (resp. s) is bounded by $R(t)$ (resp. $R'(s)$), and $A \in R$ and $B \in R'$ have the same type; and (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 the strength of a regression model $\mathcal{M}_{\text{reg}}$ and a threshold δ. We consider $\mathcal{M}$ such as (1) NLP models, *e.g.,* Bert [30], for text classification; (2) ER models and link prediction models, *e.g.,* Bert for semantic matching; and (3) models for error detection and correction, *e.g.,* generative models [113].

Predicate construction. Following the practice of [14], we construct (a) constant predicates $t.A \oplus c$ by examining all candidate values as c if A is a categorical attribute, and determining the threshold c using decision trees if A is a numerical attribute; (b) variable predicates $t.A \oplus s.B$ by identifying correlated attributes A and B via semantic embeddings; and (c) ML predicates $\mathcal{M}(t[\bar{A}], s[\bar{B}])$, by identifying correlated attributes and pre-computing the ML predictions.

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

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

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

Example 2: Consider schemas Company (cid, cname, USSC, reg_code (registration code), reg_cap (registered capital), legal_rep (legal representative), est (establishment), rev (revocation), type) and Person (pid, name, sex, ID_no.). Below are some example REEs.

(1) $\varphi_1 : \mathsf{Company}(t) \wedge \mathsf{Company}(s) \wedge t.\mathsf{est} > 2015 \wedge s.\mathsf{est} > 2015 \wedge t.\mathsf{USCC} = s.\mathsf{USCC} \rightarrow t.\mathsf{cid} = s.\mathsf{cid}$. This REE specifies φ_1 in Example 1. It identifies two companies by their USCC, issued after 2015. Similarly φ_k and φ_1' of Example 1 can be expressed as REEs.

(2) $\varphi_2 : \mathsf{Company}(t) \rightarrow t.\mathsf{est} \leq t.\mathsf{rev}$. It says that a company must be established before it is revoked; φ_2 can fix errors in attribute rev.

(3) $\varphi_3 : \mathsf{Company}(t) \wedge \mathsf{Company}(t') \wedge \mathsf{Person}(s) \wedge \mathsf{Person}(s') \wedge t.\mathsf{legal\_rep} = s.\mathsf{pid} \wedge t'.\mathsf{legal\_rep} = s'.\mathsf{pid} \wedge s.\mathsf{ID\_no.} = s'.\mathsf{ID\_no.} \wedge t.\mathsf{reg\_code} = t'.\mathsf{reg\_code} \rightarrow \mathcal{M}_{\mathsf{ER}}(t.\mathsf{cid}, t.\mathsf{cid})$, where $\mathcal{M}_{\mathsf{ER}}$ is an ER model that identifies companies t and t'. Here φ_3 explains the prediction of $\mathcal{M}_{\mathsf{ER}}$ in logic, *i.e.,* companies t and t' are identified since they have same registration code and legal representative. □

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 each variable t of φ 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 [9]. (2) If p is $\mathcal{M}(t[\bar{A}], s[\bar{B}])$, then $h \models p$ if $\mathcal{M}$ predicts true on $(h(t)[\bar{A}], h(s)[\bar{B}])$.

For precondition X, we write $h \models X$ if $h \models p$ for *all* predicates p in X. For an REE $\varphi = X \rightarrow p_0$, we write $h \models \varphi$ if $h \models X$ implies $h \models p_0$. An instance $\mathcal{D}$ of $\mathcal{R}$ *satisfies* φ, denoted by $\mathcal{D} \models \varphi$, if *for all* valuations h of tuple variables of φ in $\mathcal{D}$, $h \models \varphi$. We say that $\mathcal{D}$ *satisfies* a set Σ of REEs, denoted by $\mathcal{D} \models \Sigma$, if for all $\varphi \in \Sigma$, $\mathcal{D} \models \varphi$.

Example 3: Assume that $\mathcal{D}$ has two relations D_1 and D_2 of schema Company and Person in Tables 1

tid	cid	cname	USSC	reg_code	reg_cap	legal_rep	est	rev	type
t_1	c_1	Brilliant Tech	91310101MA1FL5Y06J	5796214-5	\$5M	p_1	2020	-	IT
t_2	c_2	Brilliant Group Holding	91310101MA1FL5Y06J	5796214-5	\$5M	p_2	2020	-	IT
t_3	c_3	Blue Sky Co., Ltd	-	2468013-7	\$2M	p_4	2010	2014	Retail
t_4	c_4	Blue Sky Co., Ltd	-	2468013-7	\$2M	p_4	2010	2014	Retail
t_5	c_5	GreenHarbor Foods	91320500MA1M9U3P5U	5796214-5	\$1M	p_5	2016	2019	Food

Table 1. Company relation D_1

tid	pid	name	sex	ID_no.
t_6	p_1	J. Smith	M	BJ2023000123456A
t_7	p_2	John Smith	M	BJ2023000123456A
t_8	p_3	J. Smith	F	SH2022005678901B
t_9	p_4	Chen Li	F	GZ2021098765432C
t_{10}	p_5	Ada Chan	F	CQ2020123456789D

Table 2. Person relation D_2

and 2, respectively. A valuation $h_1 : \{t_1 \mapsto t, t_2 \mapsto s\}$ satisfies φ_1 and it identifies c_1 and c_2. □

Properties. (1) As shown in [37], REEs subsume CFDs, DCs and MDs as special cases. More specifically, (a) DCs are REEs defined with atomic formulas of the form $t.A \oplus c$ and $t.A \oplus s.B$ on single tables. (b) CFDs are REEs defined in terms of two relation atoms $R(t)$ and $R(s)$ of the same relation R, and equality predicates $t.A = s.B$ and $t.A = c$. (c) MDs can be expressed as REEs $X \rightarrow t.\mathrm{id} = s.\mathrm{id}$ where X has two relation atoms $R_1(t)$ and $R_2(s)$, equality predicate $t.A = s.B$ and $\mathcal{M}(t[\bar{A}], s[\bar{B}])$ that simulates similarity checking. (2) As indicated in Example 2, REEs may embed ML models as predicates, and unify ER, CR and association analysis. An REE may carry multiple tuple variables for collective analysis across tables [16], *e.g.,* φ_3. (3) Moreover, one can discover logic conditions X to explain predictions of certain ML model $\mathcal{M}$ in an REE of the form $X \rightarrow \mathcal{M}(t[\bar{A}], s[\bar{B}])$, *e.g.,* φ_3, when $\mathcal{M}$ is its consequence.

3 The Discovery Problem

This section proposes a framework for rule discovery, which is open to any relevance and diversity measures. We first give example relevance and diversity measures for REEs (Section 3.1 and Section 3.2). Then we define the objective function, formulate the discovery problem for top-k relevant and diversified REEs, and show that the problem is NP-hard and hard to approximate (Section 3.3).

3.1 Relevance Measures

To discover rules that fit users' need, it is necessary to define relevance measures that capture users' prior knowledge or interests. Given an REE φ, we use $\delta_{\mathrm{rel}}(\varphi)$ to indicate the *relevance* of the rule, such that the higher $\delta_{\mathrm{rel}}(\varphi)$ is, the more relevant φ is to users' need.

Below we define a relevance measure, including (a) conventional *support* and *confidence*, and (b) an ML-based relevance model.

Support. This is to quantify how often an REE can be applied to $\mathcal{D}$. Note that our predicates can be classified into two types, namely, (a) unary predicates with only one tuple variable (*e.g.,* $t.A \oplus c$), and (b) binary predicates with two tuple variables (*e.g.,* $t.A \oplus s.B$). Following [36], we define the *support* of $\varphi : X \rightarrow p_0$ in $\mathcal{D}$, denoted by $\mathrm{supp}(\varphi, \mathcal{D})$, based on the type of consequence p_0. This is also practiced by *e.g.,* [41], which picks pivots to define the support.

(1) If p_0 is a binary predicate with tuple variables t_0 and s_0, we define $\mathrm{supp}(\varphi, \mathcal{D}) = |\mathrm{spset}(\varphi, \mathcal{D})|$, where $\mathrm{spset}(\varphi, \mathcal{D})$ is the *support set* of φ, defined as $\{\langle h(t_0), h(s_0)\rangle \mid h \text{ is a valuation of } \varphi \text{ in } \mathcal{D}, h \models X \text{ and } h \models p_0\}$ *i.e.,* the set of all tuple pairs $\langle h(t_0), h(s_0)\rangle$ with $h \models \varphi$.

(2) If p_0 is a unary predicate with variable t_0, we define $\mathrm{supp}(\varphi, \mathcal{D}) = |\mathrm{spset}(\varphi, \mathcal{D})|^2$, where $\mathrm{spset}(\varphi, \mathcal{D}) = \{h(t_0) \mid h \text{ is a valuation of } \varphi \text{ in } \mathcal{D}, h \models X \text{ and } h \models p_0\}$. Note that we take a square

on the set size so that the supports of REEs with unary consequences and the supports of REEs with binary consequences are of the same scale.

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

It is known that $\mathrm{supp}(\varphi, \mathcal{D})$ is *anti-monotonic*, *i.e.,* given $\varphi : X \rightarrow p_0$ and $\varphi' : X' \rightarrow p_0$ with the same consequence p_0, we write $\varphi \preceq \varphi'$ if $X \subseteq X'$, *i.e.,* φ is less restrictive than φ'. For any instance $\mathcal{D}$ of $\mathcal{R}$, φ and φ', if $\varphi \preceq \varphi'$, then $\mathrm{spset}(\varphi', \mathcal{D}) \subseteq \mathrm{spset}(\varphi, \mathcal{D})$ and $\mathrm{supp}(\varphi', \mathcal{D}) \leq \mathrm{supp}(\varphi, \mathcal{D})$. Similarly, we write $\varphi \prec \varphi'$ if $X \subset X'$.

Remark. We adopt the support following the common practice of rule discovery. Nevertheless, our formulation is open to any relevance measures, *e.g.,* one can define a relevance measure based on the largest sub-instance $\mathcal{D}_{\mathsf{sub}}$ of $\mathcal{D}$ such that $\mathcal{D}_{\mathsf{sub}} \models \varphi$ [70]. We can also define the support for a predicate (or a predicate set, see below) and only consider predicates above a threshold in rule discovery.

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

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

where $\mathrm{spset}(X, \mathcal{D})$ is the set of tuple pairs (resp. tuples) satisfying all predicates in X for binary consequence p_0 (resp. unary p_0). The use of confidence helps us to discover useful rules from noisy data.

For a threshold η, REE φ is *η-confident* on $\mathcal{D}$ if $\mathrm{conf}(\varphi, \mathcal{D}) \geq \eta$.

Relevance model. To capture users' prior knowledge, we learn a relevance model $\mathcal{M}_{\mathsf{rel}}$ from user data. Taking an REE φ as input, it outputs a relevance score $\mathcal{M}_{\mathsf{rel}}(\varphi)$, indicating how relevant φ is.

We learn the relevance model $\mathcal{M}_{\mathsf{rel}}$ from *relative comparisons* of a number of *rule pairs*. However, differing from [36] which merely relies on pre-trained embeddings and asks users to directly give a 0/1 label on each rule pair, our novelty includes: (a) a data-dependent rule embedding scheme and (b) a semi-automatic labeling strategy.

To learn $\mathcal{M}_{\mathsf{rel}}$, we assume the following two sets as input: (a) a set $\mathcal{D}_{\mathsf{dirty}}$ of (possible dirty) data, from which we discover a set Σ_{s} of rules for constructing the training rule pairs for the relevance model $\mathcal{M}_{\mathsf{rel}}$ and generate the data-dependent embeddings; here $\mathcal{D}_{\mathsf{dirty}}$ is sampled randomly from the original dataset $\mathcal{D}$ so that $\mathcal{D}_{\mathsf{dirty}}$ (approximately) follows the same distribution as $\mathcal{D}$; and (b) $\mathcal{D}_{\mathsf{clean}}$, the dataset $\mathcal{D}_{\mathsf{dirty}}$ cleaned by the users by employing their own methods, including rules that they have already known. Various cleaning methods are in place, *e.g.,* rule-based Holoclean [91] and ML-based Baran [72], and the users can pick whatever they need. As will be verified in Section 6, a medium size of $\mathcal{D}_{\mathsf{dirty}}/\mathcal{D}_{\mathsf{clean}}$ (*e.g.,* 20K tuples) typically suffices to provide informative rule pairs for training $\mathcal{M}_{\mathsf{rel}}$ to achieve reasonable accuracy (*e.g.,* >0.85).

Data-dependent rule embedding. Given an REE $\varphi : X \rightarrow p_0$, we create its embedding vector E_φ in a hierarchical manner.

We first generate an embedding E_p for each predicate $p \in X \cup \{p_0\}$. Then, the embedding of the precondition X, denoted by E_X, is $E_X = \rho(\sum_{p \in X} \Phi(E_p))$, where ρ and Φ are two linear layers without activation functions. Moreover, the embedding of the consequence p_0 is exactly E_{p_0}, *i.e.,* the predicate embedding of p_0. Finally, we obtain the rule embedding E_φ, by concatenating E_X and E_{p_0}.

The initial predicate embeddings play a fundamental role in the hierarchical embedding scheme. However, prior methods typically generate initial embeddings without considering data (*e.g.,* $\mathcal{D}_{\mathsf{dirty}}$) where rules are mined/applied. To tackle this, we advocate a data-dependent embedding scheme. We construct a heterogeneous multi-relational graph G to capture the connection between predicates and $\mathcal{D}_{\mathsf{dirty}}$ (see below). Then the predicate embeddings are learned via existing graph

embedding tools, *e.g.,* PyTorch-BigGraph [66].

Graph construction. The graph $G = (V, E, L)$ depicts how predicates are satisfied by $\mathcal{D}_{\text{dirty}}$, where (1) V is the set of vertices, in which each vertex represents a tuple t of $\mathcal{D}_{\text{dirty}}$ or an attribute value, (2) $E \subseteq V \times L \times V$ is the set of edges in which $e = (v, l, v')$ denotes an edge labeled with $l \in L$ from vertex v to v'. We have two types of edges in E : (a) (t, A, c), indicating that $t.A = c$; and (b) (t, p, s), indicating that the pair $(t, s) \models p$, where p is a binary predicate. Given $\mathcal{D}_{\text{dirty}}$ and a set P_{all} of predicates concerned (see below), we construct G, by iterating each tuple in $\mathcal{D}_{\text{dirty}}$ (resp. each tuple pair and each predicate) to build edges of type (a) (resp. type (b)).

Predicate set P_{all}. Given σ and η, we denote by Σ_{all} the set of all *valid* REEs, *i.e., σ*-frequent and η-confident REEs, on $\mathcal{D}$. Clearly, Σ_{all} may contain excessive rules that are irrelevant to users' applications. To reduce such rules in discovery, we adopt common strategies [35] to (1) pick some *application-dependent* candidate consequences p_0, pertaining to the application, and (2) learn a set P_{all} of predicates that we should concern from the entire space via graphical lasso [46], so that for each p in P_{all}, p is relevant to some candidate p_0.

Lightweight model with learnable bounds. We adopt a lightweight $\mathcal{M}_{\text{rel}}$ to output the relevance of a given REE φ, denoted by $\mathcal{M}_{\text{rel}}(\varphi)$. Following [36], $\mathcal{M}_{\text{rel}}$ employs a learnable bound on relevance so as to allow early termination and effective pruning in rule discovery.

$$\mathcal{M}_{\text{rel}}(\varphi) = \mathsf{UB}_{\mathsf{L}} - \mathsf{ReLU}(\mathsf{w}_{\text{light}}^{\mathsf{T}} E_{\varphi} + b_{\text{light}}),$$

where E_{φ} is rule embedding, $\mathsf{w}_{\text{light}}$, b_{light} and UB_{L} are learnable parameters and ReLU is the activation function. Note that $\mathcal{M}_{\text{rel}}(\varphi) \leq \mathsf{UB}_{\mathsf{L}}$, and thus, UB_{L} is an upper bound on the relevance score.

Labeling and training. We mine rules on $\mathcal{D}_{\text{dirty}}$ to build training rule pairs for $\mathcal{M}_{\text{rel}}$. Here any discovery method can be used, *e.g.,* level-wise mining [35] or random predicate combination [36]. We adopt the former to mine valid rules on $\mathcal{D}_{\text{dirty}}$. However, it takes $O(\Sigma_{\varphi \in C(\mathsf{P}_{\text{all}}) \times \mathsf{P}_{\text{all}}} |\mathcal{D}_{\text{dirty}}|^{|\varphi|})$ time in the worst case, where $C(\mathsf{P}_{\text{all}})$ is the power set of P_{all} and $|\varphi|$ is the number of predicates in φ. Thus we can limit the maximum number of REEs that can be mined in $\mathcal{D}_{\text{dirty}}$, from which we randomly form training rule pairs for $\mathcal{M}_{\text{rel}}$.

As it is labor-intensive for users to label a large number of rule pairs, we adopt a semi-automatic labeling strategy. Given a pair of REEs $\langle \varphi_i, \varphi_j \rangle$ mined from $\mathcal{D}_{\text{dirty}}$, we compare their relevance by comparing their ability of detecting potential errors in $\mathcal{D}_{\text{clean}}$. There are many error measures in the literature (*e.g.,* G_1, G_2 and G_3 for the number of violating pairs, the number of violating tuples and the number of tuples one has to delete to satisfy the dependency, respectively [59, 65, 70, 82]). They have different behaviors.

Specifically, given $\varphi : X \rightarrow p_0$, we assume *w.l.o.g.* that p_0 is a binary predicate with variables t_0 and s_0, and h is a valuation of φ.

(1) Measure G_1: One can regard each tuple pair $\langle h(t_0), h(s_0) \rangle$ such that $h \models X$ but $h \not\models p_0$ as an error detected by φ. Such h is called a *violation* of φ, *i.e.,* h witnesses that $\mathcal{D}_{\text{clean}} \not\models \varphi$. We write $\varphi_i \ll_{G_1} \varphi_j$ if φ_j detects more such erroneous tuple pairs than φ_i in $\mathcal{D}_{\text{clean}}$.

(2) Measure G_2: One can also regard a tuple as an error if it is involved in a violation h of φ. Similarly, we write $\varphi_i \ll_{G_2} \varphi_j$ if φ_j catches more potential erroneous tuples than φ_i in $\mathcal{D}_{\text{clean}}$.

(3) Measure G_3: The last measure is based on the minimal number of tuples to remove from $\mathcal{D}_{\text{clean}}$ for an REE to hold. Let $\mathcal{D}_i$ (resp. $\mathcal{D}_j$) be the largest sub-instance of $\mathcal{D}_{\text{clean}}$ such that $\mathcal{D}_i \models \varphi$ (resp. $\mathcal{D}_j \models \varphi$). We write $\varphi_i \ll_{G_3} \varphi_j$ if $|\mathcal{D}_i| > |\mathcal{D}_j|$ since more erroneous tuples are detected and removed from $\mathcal{D}_j$ for φ_j to hold on $\mathcal{D}_{\text{clean}}$.

As remarked in [70], G_1 and G_3 can be computed in PTIME, while G_3 can only be computed in PTIME for FDs. Nevertheless, we can reduce the computation of G_3 to a minimum vertex cover

problem, for which a 2-approximate algorithm exists, based on the concept of a *conflict graph*, in which vertices represent tuples and edges represent violations. Users are free to select any one of them. As will be shown in Section 6, we will test the effect of these error measures.

Intuitively, the potential errors detected by an REE φ in $\mathcal{D}_{\text{clean}}$ are those that are "*unknown*" and "*surprising*" to the users since $\mathcal{D}_{\text{clean}}$ has already been cleaned by their existing methods. Therefore, these errors are beyond the users' prior knowledge. The more such errors φ detects, the more "novel" and useful φ is to the users.

We adopt the Siamese neural network [20] with shared parameters to get the score of φ_i/φ_j, and train $\mathcal{M}_{\text{rel}}$ with cross entropy loss.

Example 4: Continuing with Example 1, assume that the user has cleaned $\mathcal{D}_{\text{dirty}}$ using the known rule φ_{k}. Since φ_1' is "homogeneous" to φ_{k}, it cannot detect "unknown" and "surprising" errors in $\mathcal{D}_{\text{clean}}$, while φ_1 does. Therefore, we label $\varphi_1' \ll \varphi_1$ for training $\mathcal{M}_{\text{rel}}$. □

Putting these together, we define *the relevance* of φ as

$$\delta_{\text{rel}}(\varphi) = w \cdot (\text{supp}(\varphi, \mathcal{D}) + \text{conf}(\varphi, \mathcal{D})) + (1 - w) \cdot \mathcal{M}_{\text{rel}}(\varphi),$$

where w is a weight assigned by users to the objective measure, and $\text{supp}(\varphi, \mathcal{D})$ and $\delta_{\text{rel}}(\varphi)$ are normalized in the range of $[0, 1]$.

3.2 Diversity Measures

We next propose four diversity measures $\delta_{\text{div}}(\Sigma)$ for rules. For a set Σ of REEs over schema $\mathcal{R}$, $\delta_{\text{div}}(\Sigma)$ denotes the *diversity* among the rules in Σ such that the larger $\delta_{\text{div}}(\Sigma)$ is, the more diverse the rules in Σ are. We use $\delta_{\text{div}}(\Sigma)$ to characterize the different features embedded in the rules and the coverage of the rules for a dataset.

Attribute non-overlap. Intuitively, a diverse rule set should be characterized by a variety of attributes (features). Thus, to promote diversity, a natural idea is to maximize *attribute coverage*, *i.e.,* the set of attributes covered by at least one rule in Σ. However, it raises a problem of "long" rules, that is, maximizing the attribute coverage implicitly encourages a large number of predicates in a rule's precondition, which may degrade interpretability in practice.

To rectify this, we propose *attribute non-overlap*. Informally, we sort rules in Σ as a sequence. For the i-th rule φ_i in Σ, a rule is favored if it contains less or no attributes that exist in previous $i-1$ rules. Given a rule φ, let $\text{cov}_{\text{A}}(\varphi)$ denote the set of attributes involved in φ. Then formally, we define $\delta_{\text{div}}^{\text{nonA}}(\Sigma)$ as follows:

$$\delta_{\text{div}}^{\text{nonA}}(\Sigma) = -\sum_{i=2}^{k} \left|\left(\bigcup_{j<i} \text{cov}_{\text{A}}(\varphi_j)\right) \cap \text{cov}_{\text{A}}(\varphi_i)\right|. \tag{1}$$

The measure counts the total number of overlapped attribute between each REE φ_i and all previous ones φ_j with index $j < i$.

The measure $\delta_{\text{div}}^{\text{nonA}}(\Sigma)$ turns out to be invariant to the order of rules in Σ. In other words, it is a set function, *i.e.,* $\delta_{\text{div}}^{\text{nonA}} : 2^{\Sigma_{\text{all}}} \to \mathbb{R}$, a function whose domain is a collection of sets.

Proposition 1: *The diversity measure $\delta_{\text{div}}^{\text{nonA}}(\Sigma)$ is a set function.* □

Proof: This can be proved by double counting. Fix a rule set Σ. For each attribute A, its each occurrence increases the count by 1 except for the first occurrence. Thus, $\delta_{\text{div}}^{\text{nonA}}(\Sigma)$ can be rewritten as

$$\delta_{\text{div}}^{\text{nonA}}(\Sigma) = -\sum_{A} \left(\left| \sum_{\varphi \in \Sigma} \mathbb{1}[A \text{ contained in } \varphi] - 1 \right| \right),$$

which is invariant to the order of rules in Σ. □

Predicate non-overlap. We also define a more fine-grained notion of diversity for REEs $\varphi : X \to p_0$.

We characterize *predicate non-overlap* by simply counting overlapped predicates:

$$\delta_{\text{div}}^{\text{nonP}}(\Sigma) = -\sum_{i=2}^{k} \left|\left(\bigcup_{j<i} \text{cov}_{\text{P}}(\varphi_j)\right) \cap \text{cov}_{\text{P}}(\varphi_i)\right|,$$

where $\text{cov}_{\text{P}}(\varphi) = \{p \mid p \in X\} \cup \{p_0\}$ is the set of predicates in φ.

Approximate non-overlap. The notions above treat syntactically different predicates/attributes as independent ones. However, predicates and attributes are often semantically correlated, *e.g.,* (1) *positive correlation*, *e.g.,* predicates $t.\text{age} > 18$ and $t.\text{age} > 20$; (2) *negative correlation*, *e.g.,* $t.\text{salary} \geq 10\text{K}$ and $t.\text{salary} < 10\text{K}$; (3) *taxonomy*, *e.g.,* attributes city and county. Intuitively, we do not want two rules to contain only semantically correlated predicates/attributes.

In light of this, we revise non-overlap to *approximate* non-overlap. Let $\text{Corr}(A_i, A_j)$ be the strength of correlation between attributes A_i and A_j (see more about $\text{Corr}(\cdot,\cdot)$ in [5]) and $\text{Corr}(S_i, S_j) = \sum_{A_i \in S_i, A_j \in S_j} \text{Corr}(A_i, A_j)$ between two attribute sets S_i and S_j. Then, the revised attribute non-overlap is defined as follows:

$$\delta_{\text{div}}^{\text{nonA}}(\Sigma) = -\sum_{i=2}^{k} \text{Corr}(\bigcup_{j<i} \text{cov}_{\text{A}}(\varphi_j), \text{cov}_{\text{A}}(\varphi_i)).$$

Similarly we define *approximate* non-overlap of predicates.

When the context is clear, we use $\delta_{\text{div}}^{\text{non}}$ to denote $\delta_{\text{div}}^{\text{nonA}}$ or $\delta_{\text{div}}^{\text{nonP}}$.

Tuple coverage. The notions of non-overlap aim to capture syntactic diversity in a rule set Σ, which fall short in reducing structural redundancy specific in the dataset, *e.g.,* two rules may be syntactically different but cover similar tuples. A natural remedy is to use rules that cover as many different tuples as possible, which is arguably a better goal than maximizing the sum of individual supports.

Given an REE $\varphi : X \rightarrow p_0$, we write $\text{cov}_{\text{T}}(\varphi) = \text{spset}(\varphi, \mathcal{D})$ for the coverage of φ. Then, the tuple coverage of Σ is defined as

$$\delta_{\text{div}}^{\text{cov}}(\Sigma) = |\bigcup_{\varphi \in \Sigma} \text{cov}_{\text{T}}(\varphi)|.$$

Max-Min distance over attribute/predicate. Given a set Σ of rules, another approach to measuring the diversity of Σ is to first define the pairwise distance between a pair of rules, and then leverage the classic diversity models, such as *Max-Min* diversity [52].

Denote by $\text{dis}(\varphi_i, \varphi_j)$ the distance between rules φ_i and φ_j, where dis() can be any distance function, *e.g., Jaccard Distance* or *Edit Distance* for attributes. Then, diversity $\delta_{\text{div}}^{\text{dis}}(\Sigma)$ is defined as

$$\delta_{\text{div}}^{\text{dis}}(\Sigma) = \min_{\varphi_i, \varphi_j \in \Sigma : i \neq j} \text{dis}(\varphi_i, \varphi_j).$$

This notion of diversity is especially useful when relevance depends mostly on a small number of essential attributes or predicates. In this case, $\delta_{\text{div}}^{\text{nonA}}(\cdot)$ given above may not be very helpful, as all essential attributes are likely to appear in Σ for any top-k rules.

Remark. Maximizing $\delta_{\text{div}}^{\text{dis}}(\cdot)$ may also lead to "long rules", since a rule that includes many un-selected attributes is more distant from the rest. An easy fix is to select rules of a bounded length.

Example 5: To illustrate, consider the exact attribute non-overlap as an example. Consider $\Sigma' = \{\varphi_1', \varphi_{\text{d}}\}$ and $\Sigma = \{\varphi_1, \varphi_{\text{d}}\}$, where $\varphi_{\text{d}} : \text{Company}(t) \wedge \text{Company}(s) \wedge t.\text{reg\_cap} < s.\text{reg\_cap} \rightarrow t.\text{cname} \neq s.\text{cname}$, and φ_1' and φ_1 are given in Examples 1-2. We compute $\delta_{\text{div}}^{\text{nonA}}(\Sigma')$ as $-|\text{cov}_{\text{A}}(\varphi_1') \cap \text{cov}_{\text{A}}(\varphi_{\text{d}})| = -|\{\text{cname}, \text{type}, \text{cid}\} \cap \{\text{reg\_cap}, \text{cname}\}| = -|\{\text{cname}\}| = -1$. Similarly, $\delta_{\text{div}}^{\text{nonA}}(\Sigma) = -|\text{cov}_{\text{A}}(\varphi_1') \cap \text{cov}_{\text{A}}(\varphi_{\text{d}})| = 0$. Thus, Σ is more diverse than Σ'. □

3.3 The Discovery Problem

We aim to discover rules that are as relevant to users' need as possible, and at the same time, as diverse as possible. Based on relevance $\delta_{\text{rel}}(\cdot)$ and diversity $\delta_{\text{div}}(\cdot)$ measures given earlier, we define the bi-criteria objective function $F(\cdot)$ on a set Σ of rules:

$$F(\Sigma) = \lambda \sum_{\varphi \in \Sigma} \delta_{\mathsf{rel}}(\varphi) + (1-\lambda) \cdot \delta_{\mathsf{div}}(\Sigma),$$

where $\lambda \in [0,1]$, set by users, is a configurable parameter for the trade-off between relevance and diversity. We normalize $F(\cdot)$ to a range of $[0,1]$. The normalization is straightforward for all non-negative measures, while negative measures can be turned into a non-negative one by adding a proper positive constant.

Problem statement (TopKDiv**).** We now state the top-k relevant and diversified REE discovery problem, referred to as TopKDiv.

- *Input*: Schema $\mathcal{R}$, instance $\mathcal{D}$ of $\mathcal{R}$, a positive integer k, relevance measure $\delta_{\mathsf{rel}}()$, diversity measure $\delta_{\mathsf{div}}()$, parameters λ, σ and η.
- *Output*: A set of REEs $\Sigma \subseteq \Sigma_{\mathsf{all}}$ such that (1) $|\Sigma| = k$ and (2) $F(\Sigma)$ is maximized, *i.e.,* $\Sigma = \arg\max_{\Sigma' \subseteq \Sigma_{\mathsf{all}}, |\Sigma'|=k} F(\Sigma')$.

Here Σ_{all} is the set of all valid (σ-frequent and η-confident) REEs.

New paradigm. (1) Note that *conventional rule discovery* is a special case of TopKDiv that finds all rules with only support/confidence measures. *Top-k rule discovery* [36] is also a special case without diversity ($\lambda = 1$). (2) As a framework, users may opt to provide their own functions for relevance $\delta_{\mathsf{rel}}()$ and diversity $\delta_{\mathsf{div}}()$, even a combination of multiple measures. Based on $F(\Sigma)$ defined with $\delta_{\mathsf{rel}}()$ and $\delta_{\mathsf{div}}()$, TopKDiv aims to find top-k relevant and diversified rules.

Complexity. No matter how desirable, TopKDiv is nontrivial. Recall that an algorithm is α-approximation for some $\alpha \in [0,1]$ if it always returns a solution whose objective value is at least α fraction of the optimum for a maximization problem (see [5] for proof).

Theorem 2: *The* TopKDiv *problem is* NP*-complete for all the four diversity measures. Furthermore, unless* P = NP*, for measure* $\delta_{\mathsf{div}}^{\mathsf{cov}}(\cdot)$*, it cannot be approximated beyond a factor of* $1 - 1/e$*; for measure* $\delta_{\mathsf{div}}^{\mathsf{dis}}(\cdot)$*, it cannot be approximated beyond a factor of 1/2.* □

Proof sketch: An NP algorithm simply guesses a set Σ of k REEs and checks whether $F(\Sigma) \leq c$ for a predefined bound c, in PTIME. We show that TopKDiv is NP-hard by reduction from the IndependentSet (IS) problem, the maximum k-cover problem and the Clique problem when δ_{div} is $\delta_{\mathsf{div}}^{\mathsf{non}}$, $\delta_{\mathsf{div}}^{\mathsf{cov}}$ and $\delta_{\mathsf{div}}^{\mathsf{dis}}$, respectively (see [5] for reductions). These problems are NP-complete (cf. [48]).

We show that for $\delta_{\mathsf{div}}^{\mathsf{cov}}(\cdot)$, it cannot be approximated beyond a factor of $1 - 1/e$. Consider *w.l.o.g.* rules with a single predicate. Our reduction creates a single-variable predicate p_S for each subset S in k-cover, and a tuple t_u for each element u in k-cover. Then, a tuple t_u is covered by p_S iff $u \in S$ and the hardness of $(1 - 1/e)$-approximation of the k-cover [42] directly transfers to TopKDiv.

We show that for $\delta_{\mathsf{div}}^{\mathsf{dis}}(\cdot)$, it cannot be approximated beyond a factor of 1/2. Our reduction maps every vertex in Clique to a unique REE in Σ_{all}, and a 1/2-metric over REEs as in [89] is devised for graph connectivity in Clique. Thus, a k-clique exists iff the $\delta_{\mathsf{div}}^{\mathsf{dis}}(\cdot)$ can take on a value of 2 instead 1, creating a 1/2 gap in approximation. □

4 A Discovery Algorithm for TopKDiv

We next develop an algorithm, referred to as TopKDivMiner, to discover top-k relevant and diversified rules. In light of Theorem 2, it is beyond reach in practice to expect an efficient approximate algorithm for TopKDiv. This said, we show that TopKDivMiner ensures approximation bounds under certain conditions.

One might want to first mine all rules, and then pick the top-k ones based on $F(\cdot)$. However, as indicated in the proof of Theorem 2, the decision problem of TopKDiv is at least as hard as problems that are notoriously hard [42] and require EXPTIME in k for exact optimization. Thus,

we adopt a greedy approach, which is not only sound in theory but is also practical as verified by, *e.g.,* the widely-used MMR algorithm [21]. Here we develop a conceptual TopKDivMiner and defer its pruning strategies to Section 5.

Algorithm. As shown in Algorithm 1, TopKDivMiner iteratively finds the best next φ_{next} REE *w.r.t.* previous ones in the current partial top-k set Σ, to maximize $\mathsf{score}(\varphi_{\mathsf{next}}) = F(\Sigma \cup \{\varphi_{\mathsf{next}}\}) - F(\Sigma)$. Function GreedyREE (Step 6) returns such an REE φ_{next} greedily. After k calls of GreedyREE, the set Σ of top-k REEs is returned.

Rule search. We generate candidate rules top-down. For each consequence p_0, we start with an empty precondition and gradually expand it by including predicates one by one. More specifically, suppose that the candidate REE under consideration is $\varphi : X \rightarrow p_0$, for which we maintain two predicate sets: (1) $\mathsf{P}_{\mathsf{sel}}$, the set of predicates selected to constitute X; and (2) P_{re}, the set of remaining candidate predicates. We expand X with some predicate from P_{re} to make a new REE (Step 39). Here predicates in P_{re} can be processed in different orders, *e.g.,* predicates that can constitute high score rules (resp. have high supports) are processed first to get highly ranked REEs (resp. σ-frequent REEs) early. Note that a rule needs to be expanded only if it survives our pruning (see below for pruning).

Instead of searching the entire rule space for φ_{next} in every iteration (the outer while loop at Step 5), TopKDivMiner maintains a *search boundary*, and expands every rule at most *once*. Intuitively, the boundary is a set $\mathcal{B}$ of rules such that any φ is in one of the following three cases: (a) φ is on the boundary, *i.e.,* $\varphi \in \mathcal{B}$; (b) φ is within the boundary, *i.e.,* $\varphi \prec \varphi'$ for some $\varphi' \in \mathcal{B}$, which indicates that φ has been visited before; and (c) φ is outside the boundary, *i.e.,* $\varphi' \prec \varphi$ for some $\varphi' \in \mathcal{B}$ and thus, φ has not been visited yet.

More specifically, we maintain four subsets of rules in the boundary $\mathcal{B}$: (a) $\mathcal{Q}_{\mathsf{past}}$, the rules that are promising but need no further expansion; (b) $\mathcal{Q}_{\mathsf{rm}}$, the rules that can be pruned completely and need no further expansion; (c) $\mathcal{Q}_{\mathsf{next}}$, the rules that can be pruned only for the current iteration and may be expanded in later iterations; and (d) $\mathcal{Q}$, the rules that have not yet been examined. In the current iteration, rules in $\mathcal{Q}$ will all be examined until it is empty, and the other rules within the boundary are invalid and need not be examined for a second time. Thus, the next iteration can continue the search from the current boundary outwards by letting $\mathcal{Q} := \mathcal{Q}_{\mathsf{next}}$.

Within a call to function GreedyREE, rules kept in $\mathcal{Q}_{\mathsf{past}} \cup \mathcal{Q}$ are sequentially processed. Shorter rules are prioritized for minimality (Step 16). Note that there may be newly expanded rules that are added into $\mathcal{Q}$. For each rule φ to be processed, TopKDivMiner conducts two types of tests to decide whether φ can be pruned (a) by the anti-monotonicity (Step 18), and (b) due to a low score (Steps 9 and 35). If a candidate φ survives both tests, it will be expanded to produce new REEs (Step 39).

Pruning. TopKDivMiner develops score-based pruning and multiple optimization strategies. The score of a rule is compared with two yardsticks: (1) $\mathsf{score}_{\mathsf{max}}$, which keeps track of the score of the best rule that is seen so far in the current round; any rule whose upper bound UB is below $\mathsf{score}_{\mathsf{max}}$ can be safely pruned for the current round; and (2) $\mathsf{LB}^{\mathsf{lazy}}(\varphi_i \mid \Sigma)$, where $i = k - |\Sigma|$ is the number of rounds ahead, and φ_i is the i-th rule at the search boundary by descending lower bound $\mathsf{LB}^{\mathsf{lazy}}(\cdot \mid \Sigma)$; this records a lower bound of scores of the top-k rules among rules *only* from the search boundary. Any rule whose upper bound $\mathsf{UB}^{\mathsf{lazy}}$ is below this lower bound cannot get into the final top-k. The bounds are also used to skip the actual computation of scores, which is also known as *lazy evaluation* (Step 23). We defer the details about these bounds to Section 5.

Example 6: Suppose $k = 3$, $\lambda = 0.5$ and $\mathsf{P}_{\mathsf{all}} = \{p_{\mathsf{cname}}, p_{\mathsf{type}}, p_{\mathsf{org}}, p_{\mathsf{est}}, p_{\mathsf{USCC}}, p^{2015}_{t.\mathsf{est}}, p^{2015}_{s.\mathsf{est}}, p_{\mathsf{cid}}\}$, where p_A (resp. $p^{2015}_{t.\mathsf{est}}$) is $t.A = s.A$ (resp. $t.\mathsf{est} > 2015$) and org is the organization code [4] (not shown in schema). We fix p_{cid} as consequences, assuming $\varphi_{\mathsf{k}} : p^{\approx}_{\mathsf{cname}} \wedge p_{\mathsf{type}} \rightarrow p_{\mathsf{cid}}$ is known, where $p^{\approx}_{\mathsf{cname}}$ compares similar names. We use attribute non-overlap (resp. $\mathcal{M}_{\mathsf{rel}}$) for diversity (resp. relevance).

Algorithm 1: Algorithm TopKDivMiner

Input: $\mathcal{D}$, k, $\delta_{\mathsf{rel}}(\cdot)$, $\delta_{\mathsf{div}}(\cdot)$, λ, σ and η.
Output: A set Σ of REEs s.t. $|\Sigma| = k$ and $F(\Sigma)$ is maximized

$\Sigma := \emptyset$
Initialize empty queues Q, Q_{past} and Q_{rm}
$\mathsf{P}_{\mathsf{all}}$:= the set of all predicates to be considered, with a fixed order
$\mathsf{P}_{\mathsf{sel}} := \emptyset$; $\mathsf{P}_{\mathsf{re}} := \mathsf{P}_{\mathsf{all}}$; Q.add($\langle \mathsf{P}_{\mathsf{sel}}, \mathsf{P}_{\mathsf{re}}, p_0 \rangle$) for each $p_0 \in \mathsf{P}_{\mathsf{all}}$
while $|\Sigma| < k$ **do**
  $\langle \varphi_{\mathsf{next}}, Q_{\mathsf{next}} \rangle := \mathsf{GreedyREE}(\Sigma, Q, Q_{\mathsf{past}}, Q_{\mathsf{rm}})$
  $\Sigma := \Sigma \cup \{\varphi_{\mathsf{next}}\}$; $Q := Q_{\mathsf{next}}$; $i := k - |\Sigma|$
  φ_i := the i-th valid rule in Q_{past} by descending $\mathsf{LB}^{\mathsf{lazy}}(\cdot \mid \Sigma)$
  Remove $\varphi' \in Q_{\mathsf{past}}$ s.t. $\mathsf{UB}^{\mathsf{lazy}}(\varphi') \le \mathsf{LB}^{\mathsf{lazy}}(\varphi_i \mid \Sigma)$ and $\varphi' \in Q$ s.t. $\lambda \mathsf{UB}_{\mathsf{rel}}(\varphi') + (1-\lambda) \cdot \mathsf{UB}^{\mathsf{lazy}}_{\mathsf{div}}(\varphi') \le \mathsf{LB}^{\mathsf{lazy}}(\varphi_i \mid \Sigma)$
  Add removed φ' into Q_{rm}
return Σ

Function GreedyREE $(\Sigma, Q, Q_{\mathsf{past}}, Q_{\mathsf{rm}})$**:**
  Output: The REE with the maximum marginal gain *w.r.t.* Σ
  $\mathsf{score}_{\mathsf{max}} = -\infty$; $Q'_{\mathsf{past}} := Q_{\mathsf{past}}$
  Initialize an empty queue Q_{next}
  while $Q \neq \emptyset$ *or* $Q_{\mathsf{past}} \neq \emptyset$ **do**
    Pop $Q \cup Q_{\mathsf{past}} \setminus \Sigma$ to obtain $\langle \mathsf{P}_{\mathsf{sel}}, \mathsf{P}_{\mathsf{re}}, p_0 \rangle$, smaller $\mathsf{P}_{\mathsf{sel}}$ first
    $\varphi := \mathsf{P}_{\mathsf{sel}} \rightarrow p_0$
    if $\varphi' \preceq \varphi$ *for any* $\varphi' \in Q'_{\mathsf{past}} \cup Q_{\mathsf{rm}}$ **then**
      continue
    if $\varphi' \preceq \varphi$ *for any* $\varphi' \in Q_{\mathsf{next}}$ **then**
      Q_{next}.add($\langle \mathsf{P}_{\mathsf{sel}}, \mathsf{P}_{\mathsf{re}}, p_0 \rangle$)
      continue
    if $\mathsf{UB}^{\mathsf{lazy}}(\varphi) > \mathsf{score}_{\mathsf{max}}$ **then** // See Eq. 2
      $\mathsf{score} := \lambda \delta_{\mathsf{rel}}(\varphi) + (1-\lambda) \cdot (\delta_{\mathsf{div}}(\Sigma \cup \{\varphi\}) - \delta_{\mathsf{div}}(\Sigma))$
      if $\mathsf{score} > \mathsf{score}_{\mathsf{max}}$ *and* φ *is valid* **then**
        $\mathsf{score}_{\mathsf{max}} := \mathsf{score}$; $\varphi_{\mathsf{next}} := \varphi$
    if φ *is popped from* Q_{past} **then**
      continue
    if φ *is not* σ*-frequent* **then**
      Q_{rm}.add($\langle \mathsf{P}_{\mathsf{sel}}, \mathsf{P}_{\mathsf{re}}, p_0 \rangle$)
      continue
    if φ *is valid* **then**
      Q'_{past}.add($\langle \mathsf{P}_{\mathsf{sel}}, \mathsf{P}_{\mathsf{re}}, p_0 \rangle$)
      continue
    if $\lambda \mathsf{UB}_{\mathsf{rel}}(\varphi) + (1-\lambda) \cdot \mathsf{UB}^{\mathsf{lazy}}_{\mathsf{div}}(\varphi) \le \mathsf{score}_{\mathsf{max}}$ *or* $\mathsf{UB}(\varphi \mid \Sigma) \le \mathsf{score}_{\mathsf{max}}$ **then**
      Q_{next}.add($\langle \mathsf{P}_{\mathsf{sel}}, \mathsf{P}_{\mathsf{re}}, p_0 \rangle$) // Unexpanded rules
      continue // Early termination
    for $p \in \mathsf{P}_{\mathsf{re}}$ **do**
      Q.add($\langle \mathsf{P}_{\mathsf{sel}} \cup \{p\}, \mathsf{P}_{\mathsf{re}} \setminus \{p' \mid \mathsf{id}_{p'} \le \mathsf{id}_p\}, p_0 \rangle$)
  $Q_{\mathsf{past}} := Q'_{\mathsf{past}}$
  return $\varphi_{\mathsf{next}}, Q_{\mathsf{next}}$

Itr	Rule (φ)	$\delta_{\text{rel}}(\varphi)$	$\delta_{\text{div}}(\varphi \mid \Sigma)$	score(φ)
1	$\varphi_4 : p_{\text{org}} \rightarrow p_{\text{cid}}$	0.5	0.5	1
	$p_{\text{name}} \rightarrow p_{\text{cid}}$ or $p_{\text{type}} \rightarrow p_{\text{cid}}$	0	0.5	0.5
	other singleton rules	0.4	0.5	0.9
2	$\varphi_5 : p_{\text{USCC}} \wedge p_{\text{est}} \rightarrow p_{\text{cid}}$	0.3	0.4	0.7
	$\varphi_6 : p_{\text{cname}} \wedge p_{\text{est}} \rightarrow p_{\text{cid}}$	0.2	0.4	0.6
	$\varphi_1 : p_{\text{USCC}} \wedge p^{2015}_{t.\text{est}} \wedge p^{2015}_{s.\text{est}} \rightarrow p_{\text{cid}}$	0.4	0.4	0.8
	$\varphi_1' : p_{\text{cname}} \wedge p_{\text{type}} \rightarrow p_{\text{cid}}$	0	0.4	0.4
3	$\varphi_5 : p_{\text{USCC}} \wedge p_{\text{est}} \rightarrow p_{\text{cid}}$	0.3	0	0.3
	$\varphi_6 : p_{\text{cname}} \wedge p_{\text{est}} \rightarrow p_{\text{cid}}$	0.2	0.3	0.5
	$\varphi_1' : p_{\text{cname}} \wedge p_{\text{type}} \rightarrow p_{\text{cid}}$	0	0.4	0.4

Table 3. An algorithmic example

Denote the improvement of (normalized) diversity (resp. relevance) after adding φ to Σ by $\delta_{\text{div}}(\varphi \mid \Sigma) = \delta_{\text{div}}(\Sigma \cup \{\varphi\}) - \delta_{\text{div}}(\Sigma)$ (resp. $\delta_{\text{rel}}(\varphi)$). We conduct levelwise search, starting from $\emptyset \rightarrow p_{\text{cid}}$. Suppose rules $\varphi = p \rightarrow p_{\text{cid}}$, where $p \in \mathsf{P}_{\text{all}} \setminus \{p_{\text{cid}}\}$, are added to Q and processed in order. Suppose only $\varphi_4 = p_{\text{org}} \rightarrow p_{\text{cid}}$ is valid, *i.e.,* it needs no expansion; we add it to Q_{past}. Assume other REEs have lower scores (upper bounds) than score(φ_4) (shown in Table 3). Then they are pruned and added to Q_{next}. When Q is empty, we return $\Sigma = \{\varphi_4\}$ in the 1st iteration (marked in yellow). At the end of the 1st iteration, the boundary $\mathcal{B}$ consists of all singleton rules.

The 2nd iteration begins with $Q := Q_{\text{next}}$ and rules in $Q_{\text{past}} \cup Q$ are processed. Suppose that after expansion, we have $\varphi_5 = p_{\text{USCC}} \wedge p_{\text{est}} \rightarrow p_{\text{cid}}$, $\varphi_6 = p_{\text{cname}} \wedge p_{\text{est}} \rightarrow p_{\text{cid}}$, φ_1' and φ_1, where φ_1' and φ_1 are given in Example 1, all are valid and added to Q_{past}. We examine their relevance/diversity, to decide the next rules for Σ. Note that the diversity improvement after adding either of the four is the same since they do no overlap φ_4 (except p_{cid}). However, the relevance of φ_1' is lower than the others, since it is "homogeneous" to φ_k. Let φ_1 have the highest relevance. We then compute $\Sigma = \{\varphi_4, \varphi_1\}$.

In the 3rd iteration, the next rule is selected as follows. Consider the three remaining candidates φ_1', φ_5 and φ_6 in Q_{past}. Since φ_5 and $\varphi_1 \in \Sigma$ involve the same attributes, adding φ_5 to Σ makes it undiverse. Combining this with the fact above that φ_1' is irrelevant, we thus add φ_6 to Σ and finally, $\Sigma = \{\varphi_4, \varphi_1, \varphi_6\}$. In contrast, if we select rules based on relevance alone, we may add φ_5, rather than φ_6. □

Complexity. TopKDivMiner takes at most $O(\sum_{\varphi \in C(\mathsf{P}_{\text{all}}) \times \mathsf{P}_{\text{all}}} \mathsf{T}_{\text{rel}}(\varphi) + k\,\mathsf{T}_{\text{div}}(\varphi))$ time, where $C(\mathsf{P}_{\text{all}})$ is the power set of P_{all}, $\mathsf{T}_{\text{rel}}(\cdot)$ (resp. $\mathsf{T}_{\text{div}}(\cdot)$) is the time for computing relevance (resp. diversity). Let $|\varphi|$ be the number of predicates in φ, and $|\mathcal{D}|$ be the number of tuples. Typically, $\mathsf{T}_{\text{rel}}(\varphi) = O(|\varphi||\mathcal{D}|)$, and $\mathsf{T}_{\text{div}}(\varphi) = O(|\varphi|)$ for non-overlap. For other diversity measures, $\mathsf{T}_{\text{div}}(\varphi)$ may increase to $O(k|\varphi|)$ for Max-Min Jaccard diversity or even $O(|\varphi||\mathcal{D}|)$ for tuple coverage. Note that this is the worst-case complexity; in practice our algorithm is much faster by pruning and optimization (Section 5).

Guarantees. When $\delta_{\text{div}}(\cdot)$ is $\delta^{\text{non}}_{\text{div}}$, it is beyond reach to expect an approximation bound for TopKDivMiner; as shown in Theorem 2, TopKDiv is as hard as the Independent Set problem, which has been shown to refuse any nontrivial approximation [53]. However, for other diversity measures, there exist worst-case guarantees.

Theorem 3: *When equipped with a non-negative modular relevance measure $\delta_{\text{rel}}(\cdot)$ and the tuple coverage diversity $\delta^{\text{cov}}_{\text{div}}$,* TopKDivMiner *yields a $(1-1/e)$-approximation bound for the* TopKDiv *problem.* □

Proof sketch: As a sum of modular relevance δ_{rel} and a submodular coverage function, the objective function remains a submodular function. Then a greedy algorithm like TopKDivMiner optimally maximizes such a function with $(1 - 1/e)$-approximation [77]. □

Theorem 4: *With a non-negative modular relevance measure $\delta_{\text{rel}}(\cdot)$ and the Max-Min diversity $\delta^{\text{dis}}_{\text{div}}$, by running* TopKDivMiner *twice and returning the best solution, one can obtain 4-approximation.* □

Proof sketch: The bound is achieved by running TopKDivMiner twice, with $\lambda = 1$ and $\lambda = 0$, which provably optimizes δ_{rel} and $\delta_{\text{div}}^{\text{dis}}$, respectively. The best of these two runs maintains a global 4-approximation guarantee by simple calculation (see [5]). □

5 Optimization Strategies

In this section we develop practical pruning and optimization techniques to help early termination of TopKDivMiner. We divide the pruning strategies into those within one iteration (Section 5.1), *i.e.,* a call to the function GreedyREE, and those across iterations (Section 5.2). We also develop optimization strategies for tuple coverage and parallelization with performance guarantee (Section 5.3).

5.1 Pruning within One Greedy Iteration

Recall that our rule discovery is conducted in a top-down manner. During the top-down rule expansion in each greedy iteration, we prune future unseen REEs that are unlikely to become the best REE in the current iteration, which reduces the search space.

Given a partial top-k set Σ, we hope to find the next best REE φ_{next} that maximizes marginal score$(\varphi_{\text{next}}) = F(\Sigma \cup \{\varphi_{\text{next}}\}) - F(\Sigma)$. The idea in our pruning strategies is to compute an upper bound UB for any given REE candidate φ, such that the marginal score of any expanded REE from φ is no more than UB. That is, $\text{UB}(\varphi \mid \Sigma) \geq \max_{\varphi \preceq \varphi'} \text{score}(\varphi')$. We maintain the best REE seen so far in the current iteration, and if its marginal score is at least $\text{UB}(\varphi \mid \Sigma)$, we can safely skip φ and any expanded REE from φ in this iteration.

Like the objective function F, the upper bound UB is composed of upper bounds UB_{rel} for relevance and UB_{div} for diversity, *i.e.,*

$$\text{UB}(\varphi \mid \Sigma) = \lambda \cdot \text{UB}_{\text{rel}}(\varphi) + (1 - \lambda) \cdot \text{UB}_{\text{div}}(\varphi \mid \Sigma), \quad \text{where}$$
$$\text{UB}_{\text{rel}}(\varphi) \geq \max_{\varphi \preceq \varphi'} \delta_{\text{rel}}(\varphi'), \quad \text{UB}_{\text{div}}(\varphi \mid \Sigma) \geq \max_{\varphi \preceq \varphi'} \delta_{\text{div}}(\varphi' \mid \Sigma).$$

Here $\delta_{\text{div}}(\varphi \mid \Sigma) = \delta_{\text{div}}(\Sigma \cup \{\varphi\}) - \delta_{\text{div}}(\Sigma)$ is the marginal gain in diversity. We now show how to compute $\text{UB}(\varphi \mid \Sigma)$ for each measure.

[P1] *Relevance (support).* Adding predicates to the precondition of a rule φ does not increase its support by the *anti-monotonicity*, *i.e.,* $\text{supp}(\varphi', \mathcal{D}) \leq \text{supp}(\varphi, \mathcal{D})$ for any $\varphi' \succeq \varphi$. Thus, the support of φ can be directly used as an upper bound, *i.e.,* $\text{UB}_{\text{rel}}(\varphi) = \text{supp}(\varphi, \mathcal{D})$.

[P2] *Relevance (confidence).* The confidence is within $[0, 1]$, but is neither monotonic nor anti-monotonic. Thus we set $\text{UB}_{\text{rel}}(\varphi) = 1$.

[P3] *Relevance (model).* We can use the learned UB_{L} as an upper bound on relevance score $\mathcal{M}_{\text{rel}}(\cdot)$. Thus $\text{UB}_{\text{rel}}(\cdot) = \text{UB}_{\text{L}}$ for REE φ.

[P4] *Diversity (non-overlap).* Expanding the precondition of an REE φ with more predicates will only increase the overlap with previous REEs in Σ, and decrease the non-overlap measure. Thus, the upper bound UB_{div} is set as $\text{UB}_{\text{div}}(\varphi \mid \Sigma) = \delta_{\text{div}}(\varphi \mid \Sigma)$.

[P5] *Diversity (tuple coverage).* Similar to the notion of support, one can verify that tuple coverage is also *anti-monotonic*. Therefore, the upper bound UB_{div} is simply $\text{UB}_{\text{div}}(\varphi \mid \Sigma) = \delta_{\text{div}}(\varphi \mid \Sigma)$.

[P6] *Diversity (Max-Min distance).* The Max-Min measure is nonincreasing in an REE set, *i.e.,* $\delta_{\text{div}}(\Sigma) \geq \delta_{\text{div}}(\Sigma')$ for any $\Sigma' \supseteq \Sigma$. Thus, the marginal gain of any rule φ must be non-positive, and one could set $\text{UB}_{\text{div}}(\cdot) = 0$. A tighter upper bound is $\text{UB}_{\text{div}}(\varphi \mid \Sigma) = \max_{\varphi' \in \Sigma_{\text{all}}} \delta_{\text{div}}(\Sigma \cup \{\varphi'\}) - \delta_{\text{div}}(\Sigma)$, which is equivalent to the *farthest string* problem if we adopt the edit distance as the metric [64]. More specifically, an REE φ can be seen as a 0/1 string of length $|\text{P}_{\text{all}}|$, where each 1 indicates the appearance of one predicate in φ.

If multiple measures are used in δ_{rel} together by summation, a sum of their bounds serves as a new bound of δ_{rel}; similarly for δ_{div}.

5.2 Pruning across Greedy Iterations

To prune across greedy iterations, we study upper and lower bounds of $\delta_{\text{div}}(\varphi \mid \Sigma)$ that will hold despite changes to φ and Σ. In contrast, the bounds in Section 5.1 hold only for a fix partial rule set Σ.

We adapt a *lazy evaluation* acceleration technique to rule discovery, which has been studied for submodular maximization [75]. In TopKDivMiner, every time the current partial REE set Σ is updated, the marginal gain of all the other REEs φ in the diversity measures, *i.e.,* $\delta_{\text{div}}(\varphi \mid \Sigma)$, may change and need updating. This incurs excessive computation. The original idea of lazy evaluation is to skip such updates whenever possible. Below we not only leverage this acceleration technique to reduce unnecessary computation, but also extend it further for pruning across greedy iterations.

Upper bounds. Lazy evaluation works well when the measure presents a "*diminishing returns*" property. That is, as the partial set Σ grows, the marginal gain of any rule φ is nonincreasing. An example for this property is tuple coverage; as more rules are added to Σ, the marginal coverage of a rule φ never increases. That is, $\delta_{\text{div}}(\varphi \mid \Sigma) \geq \delta_{\text{div}}(\varphi \mid \Sigma')$ for any $\Sigma' \supseteq \Sigma$. In light of this, the marginal gain $\delta_{\text{div}}(\varphi \mid \Sigma_{\text{prev}})$ of φ from any *previous* greedy iteration such that $\Sigma_{\text{prev}} \subseteq \Sigma$ can be directly used as an upper bound $\mathsf{UB}^{\text{lazy}}_{\text{div}}$. Thus let

$$\mathsf{UB}^{\text{lazy}}(\varphi) = \lambda \cdot \delta_{\text{rel}}(\varphi) + (1-\lambda) \cdot \mathsf{UB}^{\text{lazy}}_{\text{div}}(\varphi) \tag{2}$$

be an upper bound on the marginal score of φ. If $\mathsf{UB}^{\text{lazy}}(\varphi)$ is no greater than that of the best candidate REE seen so far in the current iteration, there is no need to evaluate the true gain $\delta_{\text{div}}(\varphi \mid \Sigma)$ of φ.

A *nonincreasing* measure also goes along with the lazy evaluation technique, as the marginal gain of any φ is non-positive, and one can simply let $\mathsf{UB}^{\text{lazy}}(\varphi) = \lambda\delta_{\text{rel}}(\varphi)$. That is, we set $\mathsf{UB}^{\text{lazy}}_{\text{div}}(\varphi) = 0$.

It is easy to see that all diversity measures we propose in Section 3.2 satisfy one of the aforementioned properties.

Proposition 5: *The non-overlap, tuple coverage measures satisfy the "diminishing returns" property* w.r.t. *the partial rule set Σ, while the Max-Min distance measure is nonincreasing in Σ.* □

Proof: Coverage functions are examples for the "diminishing returns" property, and thus it is true for the tuple coverage measure. The case for the Max-Min distance is also obvious by definition.

As for the non-overlap measure, we demonstrate with attribute non-overlap; similarly for predicate non-overlap. When more rules are added to Σ, the total number of attributes involved in Σ is nondecreasing, and so is the overlap of a new φ with Σ. As the marginal gain of φ is the negation of its overlap, it is nonincreasing. □

Lower bounds. By symmetry, a lower bound $\mathsf{LB}^{\text{lazy}}$ of a rule φ is:

$$\mathsf{LB}^{\text{lazy}}(\varphi \mid \Sigma) = \lambda \cdot \delta_{\text{rel}}(\varphi) + (1-\lambda) \cdot \mathsf{LB}^{\text{lazy}}_{\text{div}}(\varphi \mid \Sigma), \tag{3}$$

where $\mathsf{LB}^{\text{lazy}}_{\text{div}}(\varphi \mid \Sigma) \leq \delta_{\text{div}}(\varphi \mid \Sigma')$ for any $\Sigma' \supseteq \Sigma$. Note that we make the conditioned Σ explicit as the lower bound is used only for pruning, not for lazy evaluation. Some options for $\mathsf{LB}^{\text{lazy}}_{\text{div}}$ are:

- attribute non-overlap: $\mathsf{LB}^{\text{lazy}}_{\text{div}}(\varphi \mid \Sigma) = -2|\varphi|$,
- predicate non-overlap: $\mathsf{LB}^{\text{lazy}}_{\text{div}}(\varphi \mid \Sigma) = -|\varphi|$,
- tuple coverage: $\mathsf{LB}^{\text{lazy}}_{\text{div}}(\varphi \mid \Sigma) = 0$, and
- Max-Min edit/Jaccard distance: $\mathsf{LB}^{\text{lazy}}_{\text{div}}(\varphi \mid \Sigma) = -\delta^{\text{dis}}_{\text{div}}(\Sigma)$.

Here $|\varphi|$ is the length of the REE φ. Most bounds above are straightforward, and thus we briefly explain the last one. The Max-Min measure may drop to zero among distinct rules, because it is possible for two distinct rules to use the same set of attributes. This is also true over predicates, as two

distinct rules may use the same set of predicates with different predicates being their consequences.

Pruning across iterations. Note that $\mathsf{UB}^{\mathsf{lazy}}_{\mathsf{div}}$ differs from $\mathsf{UB}_{\mathsf{div}}$ above; the former holds *w.r.t.* a varying Σ, *i.e.,* $\mathsf{UB}^{\mathsf{lazy}}_{\mathsf{div}}(\varphi) \geq \delta_{\mathsf{div}}(\varphi \mid \Sigma')$ for any future partial set $\Sigma' \supseteq \Sigma$, while the latter holds *w.r.t.* a varying φ, *i.e.,* $\mathsf{UB}_{\mathsf{div}}(\varphi \mid \Sigma) \geq \delta_{\mathsf{div}}(\varphi' \mid \Sigma)$ for any expanded rule $\varphi' \succeq \varphi$. This said, it happens to be the case that all $\mathsf{UB}^{\mathsf{lazy}}_{\mathsf{div}}$'s we propose are also valid for $\mathsf{UB}_{\mathsf{div}}$, *i.e.,* $\mathsf{UB}^{\mathsf{lazy}}_{\mathsf{div}}(\varphi) \geq \mathsf{UB}_{\mathsf{div}}(\varphi \mid \Sigma)$. In other words, $\mathsf{UB}^{\mathsf{lazy}}_{\mathsf{div}}$ holds *w.r.t.* varying φ and Σ simultaneously; as a consequence, we can use $\mathsf{UB}^{\mathsf{lazy}}_{\mathsf{div}}$ for cross-iteration pruning.

Given a partial set Σ, the idea of cross-iteration pruning is to maintain a lower bound LB on the contribution of the next $k - |\Sigma|$ rules to be added to Σ. Let $i = k - |\Sigma|$. We set LB to be $\mathsf{LB}^{\mathsf{lazy}}(\varphi_i \mid \Sigma)$, where φ_i is the i-th rule among valid candidates that have been explored by descending $\mathsf{LB}^{\mathsf{lazy}}(\cdot \mid \Sigma)$. Then, a rule φ and its unseen derivatives can be repeatedly pruned as long as $\mathsf{UB}^{\mathsf{lazy}}_{\mathsf{div}}(\varphi) \leq \mathsf{LB}$.

5.3 More Optimizations

We develop strategies for tuple coverage and parallelization. For the lack of space, we give outlines below and defer the details to [5].

Fast Tuple Coverage Computation. Tuple coverage poses computational challenges to REE discovery. Given a partial set Σ, an inevitable step for it is to compute the marginal coverage $\mathsf{cov}_{\mathsf{T}}(\varphi \mid \Sigma)$ of another rule φ, where $\mathsf{cov}_{\mathsf{T}}(\varphi \mid \Sigma) = \mathsf{cov}_{\mathsf{T}}(\varphi) \backslash \mathsf{cov}_{\mathsf{T}}(\Sigma)$, and $\mathsf{cov}_{\mathsf{T}}(\Sigma) = \cup_{\varphi' \in \Sigma} \mathsf{cov}_{\mathsf{T}}(\varphi')$. Recall that the coverage of an REE with a binary consequence is defined over the space of *tuple pairs*, which takes quadratic time in the number of tuples and is costly. Thus we want to avoid materializing tuple pairs when computing the union coverage of Σ and its intersection with that of φ. We devise a fast unbiased estimation for $|\mathsf{cov}_{\mathsf{T}}(\varphi \mid \Sigma)|$ via *Monte-Carlo sampling*.

Parallelization of the Algorithm. To scale with large datasets, we parallelize TopKDivMiner to be PTopKDivMiner.

Following [62], we measure the effectiveness of parallel algorithms $\mathcal{A}_p$ using the notion of parallel scalability. We say that $\mathcal{A}_p$ is *parallelly scalable* relative to a sequential algorithm $\mathcal{A}$ if its worst-case running time $T(n)$ by using p processors can be expressed as:

$$T(n) = O(t(n)/p) + p^{O(1)},$$

where $t(n)$ is the worst-case running time of $\mathcal{A}$. Here the problem instance size n includes the size $|\mathcal{D}|$ of datasets and the number of predicates ($|\mathsf{P}_{\mathsf{all}}|$), the dominating factors of problem TopKDiv.

Theorem 6: PTopKDivMiner *is parallelly scalable relative to sequential* TopKDivMiner. □

6 Experimental Study

Using real-life data, we experimentally evaluated (1) the (parallel) scalability of our algorithm for discovering top-k relevant and diversified REEs, and the effectiveness of (2) our relevance model and (3) diversity measures, and (4) our bi-criteria objective function.

Experimental setting. We start with our experimental settings.

Datasets. We used six real-life datasets $\mathcal{D}$ (Table 4): Adult, Airport, Hospital, Inspection and NCVoter are commonly used in previous studies, and DBLP is an academic dataset with multiple relations. Note that rule discovery was conducted on the entire dataset $\mathcal{D}$.

For each $\mathcal{D}$, we sampled 20K tuples of $\mathcal{D}$, denoted by $\mathcal{D}_{\mathsf{dirty}}$. We cleaned $\mathcal{D}_{\mathsf{dirty}}$ by the state-of-the-art method Holoclean [91] and got another set $\mathcal{D}_{\mathsf{clean}}$. Here $\mathcal{D}_{\mathsf{dirty}}$ and $\mathcal{D}_{\mathsf{clean}}$ were used to train the relevance model $\mathcal{M}_{\mathsf{rel}}$. In addition, we evaluated the quality of the REEs discovered by manually inspecting the real errors in $\mathcal{D}_{\mathsf{clean}}$ caught by the rules. We used small sampled

Name	Type	#tuples	#attributes	#relations
Adult [61, 70, 83]	real-life	32,561	15	1
Airport [70, 83]	real-life	55,113	18	1
Hospital [17, 26, 70]	real-life	114,919	15	1
Inspection [70, 91]	real-life	220,940	17	1
NCVoter [70, 83]	real-life	1,681,617	12	1
DBLP [104]	real-life	1,799,559	18	3

Table 4. Dataset statistic

$\mathcal{D}_{\text{clean}}$ such that we could manually confirm real errors in it. Intuitively, we assumed that the errors corrected by Holoclean in $\mathcal{D}_{\text{clean}}$ are already known. The errors caught in $\mathcal{D}_{\text{clean}}$ are unknown/surprising to the users.

ML models. We used two ML predicates in REEs: ditto [67] for ER, and Bert [90] for textual attributes. For our relevance model $\mathcal{M}_{\text{rel}}$, we utilized PyTorch-BigGraph [66] to initialize predicate embeddings and set the size to 100. Predicate embeddings are also learned during training. We used 2 hidden layers and set their dimension to 100. We employed Adam optimizer with a batch-size of 128; the learning rate is 0.001. The default error measure is G_2, following the setting in [38]. To train $\mathcal{M}_{\text{rel}}$, we limited the maximum number of rules mined on $\mathcal{D}_{\text{dirty}}$ to be 1,000, from which we randomly selected 1,000 rule pairs. The rule pairs were divided into training and testing data with 8:2 ratio. We trained $\mathcal{M}_{\text{rel}}$ with 400 epochs on training data, and evaluated it on testing data. The inference of $\mathcal{M}_{\text{rel}}$ in discovery is re-implemented with EJML library [1].

Baselines. We implemented PTopKDivMiner in Java; it adopts support, confidence, $\mathcal{M}_{\text{rel}}$ and the combination of all diversity measures in Section 3. We tested: (1) PTopk-Miner [36], a top-k rule discovery method, which has its own relevance model and does not consider diversity. (2) REEFinder [35], an exhaustive method that discovers the entire set Σ_{all} of REEs. (3) DCFinder [26], the state-of-the-art method that mines all DCs; we parallelize it for a fair comparison. (4) Random, which randomly returns k rules from Σ_{all}.

We also evaluated the following variants: (4) $\text{PTopKDivMiner}_{\text{AN}}$, $\text{PTopKDivMiner}_{\text{PN}}$, $\text{PTopKDivMiner}_{\text{AD}}$, $\text{PTopKDivMiner}_{\text{TC}}$, four variants of PTopKDivMiner that only use attribute non-overlap, predicate non-overlap, max-min attribute distance and tuple coverage as the diversity measure, respectively. (5) $\text{PTopKDivMiner}_{\text{NoDiv}}$ (resp. $\text{PTopKDivMiner}_{\text{NoRel}}$), a variant that does not apply any diversity (resp. relevance) measure. Note that $\text{PTopKDivMiner}_{\text{NoDiv}}$ does not need to update the diversity scores of REEs at each greedy iteration; it can return top-k rules in one iteration, by maintaining a heap on relevance scores. (6) $\text{PTopKDivMiner}_{\text{nop}}$, a variant without any pruning strategy, *i.e.,* it mines all rules and greedily returns the top-k. We compared with the variants in (4), (5) and (6) to verify the effect of *each* diversity measure, to justify the need for both relevance and diversity, and testing the pruning strategies, respectively.

Moreover, to test the accuracy of our relevance model $\mathcal{M}_{\text{rel}}$ we compared: (1) Bert [30], a language model (distilbert-base-uncased) fine-tuned to compare rule relevance, (2) $\mathcal{M}_{\text{sub}}$ [36], the subjective model in PTopk-Miner, (3) $\mathcal{M}_{\text{rel}}^{\text{Bert}}$, a variant of $\mathcal{M}_{\text{rel}}$ with predicate embeddings initialized by Bert [30], (4) $\mathcal{M}_{\text{rel}}^{G_1}$ and (5) $\mathcal{M}_{\text{rel}}^{G_3}$, two variants of $\mathcal{M}_{\text{rel}}$ that use G_1 and G_3 as the error measure, respectively.

Metrics. For each method, we evaluated its scalability by the discovery time on $\mathcal{D}$ and the effectiveness by both the relevance/diversity scores and the accuracy of (unknown) error detection. Consider a set Σ of k rules returned by a method and the entire set Σ_{all} from $\mathcal{D}$.

(1) We evaluated Σ by normalized relevance and diversity scores, computed as $F_{\text{rel}} = \sum_{\varphi \in \Sigma} \mathcal{M}_{\text{rel}}(\varphi)$ and $F_{\text{div}} = \delta_{\text{div}}^{\text{nonA}}(\Sigma) + \delta_{\text{div}}^{\text{nonP}}(\Sigma) + \delta_{\text{div}}^{\text{cov}}(\Sigma) + \delta_{\text{div}}^{\text{dis}}(\Sigma)$ (*i.e.,* all diversity measures), respectively.

(2) To evaluate the practical effectiveness of our relevance and diversity measures, we examined the errors detected in the following three ways. Denote by $\mathsf{Vio}(\Sigma)$ (resp. $\mathsf{Vio}(\varphi)$) the set of all errors caught as the violations of all rules in Σ (resp. a single rule φ).

(2a) Real errors in $\mathcal{D}_{\mathsf{clean}}$ in terms of Precision, Recall and F1 defined as $2 \cdot \mathsf{Precision} \cdot \mathsf{Recall}/(\mathsf{Precision} + \mathsf{Recall})$, where Precision (resp. Recall) is the ratio of detected *real errors* to all the violations of REEs in Σ (resp. all real errors). Intuitively, the more unknown errors detected in $\mathcal{D}_{\mathsf{clean}}$, the more relevant the rules are.

(2b) Error coverage in $\mathcal{D}_{\mathsf{clean}}$, *i.e.,* the ratio of violations of rules in the top-k set Σ to the violations of the set Σ_{all} of all rules, denoted by $\mathsf{ECratio} = \frac{|\mathsf{Vio}(\Sigma)|}{|\mathsf{Vio}(\Sigma_{\mathsf{all}})|}$. Intuitively, errors caught by homogeneous rules tend to coincide. In contrast, an effective diversity measure promotes diverse top-k rules, which should be able detect different cases of errors compared to those caught by the entire set Σ_{all}.

(2c) Error dissimilarity in $\mathcal{D}_{\mathsf{clean}}$, *i.e.,* the average Jaccard distance between the violations of every distinct rule pair in Σ, denoted by $\mathsf{JD} = \frac{1}{|\Sigma|(|\Sigma|-1)} \sum_{\varphi_i,\varphi_j \in \Sigma, \varphi_i \neq \varphi_j} \left(1 - \frac{|\mathsf{Vio}(\varphi_i) \cap \mathsf{Vio}(\varphi_j)|}{|\mathsf{Vio}(\varphi_i) \cup \mathsf{Vio}(\varphi_j)|}\right)$. Intuitively, the more dissimilar the errors caught, the more diverse the rules are.

We also evaluated the usefulness of rules (*i.e.,* whether they are relevant and diverse to real users), by setting up 10 independent error detection tasks over five datasets; each task aims to detect certain cases of errors, *e.g.,* a task on Adult detects erroneous salary of users and another task on Inspection detects a wrong level of risk. We report the average performance over these 10 tasks. For each task, we mined three lists of top-10 REEs using (a) PTopKDivMiner, (b) PTopk-Miner and (c) Random. We asked domain experts, who understand the purpose of the task, to give two sets of labels, for evaluating the relevance and diversity of the rules, respectively.

(1) Relevance. The 30 rules (*i.e.,* 10 from each list) are shuffled and presented to the experts. To facilitate labeling, for each rule, we randomly selected 10 errors it detects and complemented each rule with the errors. A rule was labeled 1 if the expert found the detected errors useful, *e.g.,* unknown/surprising to the task. Then we compared the number of relevance labels received by each method; the more labels a method receives, the more relevant the rules are.

(2) Diversity. We randomly selected 20 pairs of rules from each list, yielding 60 rule pairs in total. These rule pairs are shuffled and presented to the experts. A rule pair was labeled 1 if the expert found the errors detected by the two rules are dissimilar. The more labels a method receives, the more diverse the rules are.

Configuration. We conducted experiments on a cluster of up to 21 virtual machines, each powered by 64GB RAM and 16 processors with 2.20 GHz. We ran the experiments 3 times, and report the average here. Unless stated explicitly, we set the number of machines $n = 20$, the support threshold $\sigma = 10^{-6} \cdot |\mathcal{D}|^2$, the confidence $\eta = 0.75$, $k = 10$ in top-k discovery, and the trade-off parameter $\lambda = \frac{1}{3}$.

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

Exp-1: Scalability. We tested the scalability, varying the (1) value k, (2) support σ, (3) confidence η, (4) the number n of machines, (5) the size of $\mathcal{D}$, (6) the size of $|X|$ and (7) the diversity measures.

Varying k. We varied k from 1 to 40 in Figures 1(a) and 1(b). (1) REEFinder, DCFinder and $\mathsf{PTopKDivMiner}_{\mathsf{nop}}$ are not sensitive to k, since they either mine all REEs/DCs, regardless of k, or apply no pruning. In contrast, PTopKDivMiner takes much less time; on average, it is 62.4X, 51.5X and 15.02X faster than the three, respectively. This speedup is particularly evident when k is small, *e.g.,* it is 51.66X faster than $\mathsf{PTopKDivMiner}_{\mathsf{nop}}$ when $k = 1$, justifying the need of

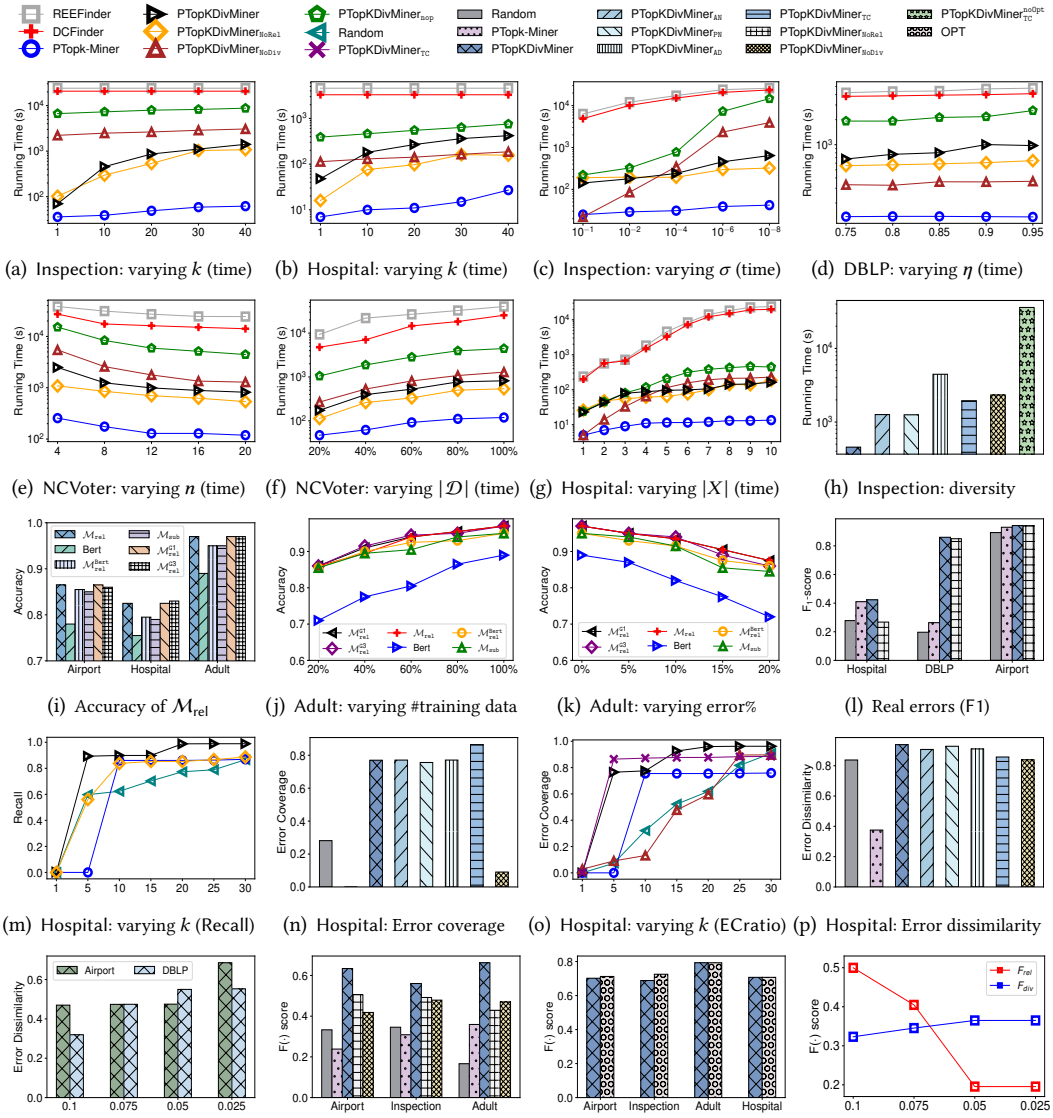

Fig. 1. Performance evaluation

effective pruning strategies. (2) It is comparable to both its variants PTopKDivMiner$_{\text{NoDiv}}$ and PTopKDivMiner$_{\text{NoRel}}$, although it incorporates both relevance and diversity in discovery. In particular, the time of PTopKDivMiner$_{\text{NoDiv}}$ only increases slightly with k since (a) it is a single-iteration variant and (b) its top-k pruning with only relevance bounds is less effective, *e.g.*, for all REEs φ, the bounds on $\mathcal{M}_{\text{rel}}(\varphi)$ are the same (*i.e.*, UB_{L}). (3) PTopKDivMiner takes longer when k gets larger, as expected, since more REEs are checked. Although PTopKDivMiner is slower than PTopk-Miner, its rules are both relevant and diverse, catching more (unknown) errors. We defer the discussion about this to Exp-2 and Exp-3.

Varying support. Decreasing σ from $10^{-1}|\mathcal{D}|^2$ to $10^{-8}|\mathcal{D}|^2$ in Figure 1(c), all algorithms take longer when σ is smaller, as expected, since they need to examine more candidates, *e.g.,* PTopKDivMiner is 4.5X slower when σ changes from $10^{-1}|\mathcal{D}|^2$ to $10^{-8}|\mathcal{D}|^2$. Nevertheless, it is consistently faster than PTopKDivMiner$_{\text{nop}}$, REEFinder and DCFinder in all cases, *e.g.,* 9.08X, 55.51X and 47.24X faster than the three on average, respectively, up to 22.87X, 72.25X and 63.14X.

Varying confidence. Increasing η from 0.75 to 0.95 in Figure 1(d), most algorithms are faster given a smaller η, *e.g.,* PTopKDivMiner is 1.45X faster when η is from 0.95 to 0.75. This is because higher η requires to check more candidate REEs. Nonetheless, it still beats PTopKDivMiner$_{\text{nop}}$, REEFinder and DCFinder in all the cases.

Varying n. Varying the number n of machines from 4 to 20, we evaluated the parallel scalability of PTopKDivMiner. As shown in Figure 1(e), PTopKDivMiner is 3.05X faster when n varies from 4 to 20. It is feasible in practice; it takes 818s on NCVoter when $n = 20$.

Varying $|\mathcal{D}|$. Using NCVoter, one of the largest datasets with 1.68M tuples, we tested the impact of the size $|\mathcal{D}|$ by varying the scaling factor from 20% to 100%, *i.e.,* the number of tuples is changed from 20% to 100%. As shown in Figure 1(f), all the algorithms take longer, as expected. This said, PTopKDivMiner outperforms PTopKDivMiner$_{\text{nop}}$, REEFinder and DCFinder in all the cases.

Varying $|X|$. We varied the maximum number of predicates in X in Figure 1(g). As shown there, the post-processing approaches, *e.g.,* PTopKDivMiner$_{\text{nop}}$ that mines all rules and greedily returns the top-k, exhibit a steep rise in time with increasing $|X|$, while the time of PTopKDivMiner only increases slightly when $|X|$ gets larger.

Diversity. We evaluated the impact of diversity measures on PTopKDivMiner, PTopKDivMiner$_{\text{NoDiv}}$, PTopKDivMiner$_{\text{AN}}$, PTopKDivMiner$_{\text{PN}}$, PTopKDivMiner$_{\text{AD}}$ and PTopKDivMiner$_{\text{TC}}$. To evaluate the optimization strategy for fast tuple coverage (see Section 5.3), we used an additional variant of PTopKDivMiner$_{\text{TC}}$, denoted by PTopKDivMiner$_{\text{TC}}^{\text{noOpt}}$, without the optimization. As shown in Figure 1(h), (a) although PTopKDivMiner$_{\text{NoDiv}}$ only runs one iteration and other methods run k greedy iterations, incorporating diversity measures takes less time than PTopKDivMiner$_{\text{NoDiv}}$ (except PTopKDivMiner$_{\text{AD}}$), due to the effective upper bounds on all diversity measures. (b) By combining all the diversity measures, PTopKDivMiner achieves the most significant speedup, since all upper bounds are able to apply for pruning, *e.g.,* PTopKDivMiner is 5.11X faster than PTopKDivMiner$_{\text{NoDiv}}$. (c) Among all diversity measures, max-min attribute distance is the slowest, since its upper bound is not as tight as the others, which presents the "diminishing returns" property (Section 5.2). (d) Our optimization for tuple coverage is effective, without which PTopKDivMiner$_{\text{TC}}^{\text{noOpt}}$ takes >10h.

Exp-2: Effectiveness of relevance model. We evaluated the accuracy of $\mathcal{M}_{\text{rel}}$, based on the metrics given above for relevance.

Accuracy of relevance. We first compared the accuracy of $\mathcal{M}_{\text{rel}}$ with Bert, $\mathcal{M}_{\text{sub}}$ and its variants; here the accuracy was measured by the percentage of rule pairs in testing data whose relative rank is correctly identified. As shown in Figure 1(i), $\mathcal{M}_{\text{rel}}$ beats Bert and $\mathcal{M}_{\text{sub}}$ on all datasets, *e.g.,* its accuracy is 9.72% and 2.77% higher than Bert and $\mathcal{M}_{\text{sub}}$ on average, respectively, up to 10.9% and 4.43%. Besides, $\mathcal{M}_{\text{rel}}$ is on average 2.35% more accurate than $\mathcal{M}_{\text{rel}}^{\text{Bert}}$; this verifies the usefulness of our data-dependent embedding strategy, which takes the underlying data regularity into consideration. Moreover, the accuracy of $\mathcal{M}_{\text{rel}}$ is comparable to $\mathcal{M}_{\text{rel}}^{G_1}$ and $\mathcal{M}_{\text{rel}}^{G_3}$, thereby confirming that none of them can be unequivocally regarded as the best one.

Varying #training. We varied the number of training rule pairs for relevance models in Figure 1(j) from 20% to 100%. As expected, all methods get more accurate given more training data, *e.g.,* with

40% training pairs, the accuracy of $\mathcal{M}_{\text{rel}}$ is as high as 0.895, indicating that a medium-size training set suffices to get reasonable accuracy.

Varying error%. We also tested the impact of the quality of training data. Given an error parameter β%, we randomly selected β% of training pairs and flipped their labels. Varying the β% from 0% to 20%, we reported the accuracy of all relevance models in Figure 1(k). With more erroneous labels, the accuracy decreases, *e.g.,* the accuracy of $\mathcal{M}_{\text{rel}}$ drops from 0.97 to 0.875. Nonetheless, $\mathcal{M}_{\text{rel}}$ is more robust to errors than the other relevance models.

Accuracy of real error detection. We examined real errors in $\mathcal{D}_{\text{clean}}$ caught by rules of PTopKDivMiner, PTopKDivMiner$_{\text{NoRel}}$, PTopk-Miner and Random. As shown in Figure 1(l), Random performs the worst since randomly selected rules can hardly fit the user's need. In contrast, PTopKDivMiner performs the best: its average F1 is 0.74, as opposed to 0.68 and 0.53 by PTopKDivMiner$_{\text{NoRel}}$ and PTopk-Miner, respectively. The advantage of PTopKDivMiner is particularly evident on DBLP since the rules from PTopk-Miner only detect errors on a few attributes (*i.e.,* homogeneous rules), leading to low recall. Note that all methods have lower F1 on Hospital. This is because Holoclean has done a better job on the dataset that has more duplicates than the others, and it is harder to find errors missed.

We also tested the impact of quantity/quality of training data on real errors caught by PTopKDivMiner, in the the same setting as in Figures 1(j)-1(k) (not shown). When the number of training data changes from 20% to 100% on DBLP, F1 increases from 0.61 to 0.86. Moreover, when β% changes from 0% to 20% on Hospital, F1 slightly drops by 0.07. This verifies that PTopKDivMiner is also robust to errors and performs well with a reasonable amount of training data.

Statistical test on real error detection. To test whether the real error F1 of PTopKDivMiner is significantly different from PTopk-Miner, we conducted Student's t-test [79] with significance level as 0.05. We randomly drew 100 sample datasets $\mathcal{D}_s$ from $\mathcal{D}$, where each $|\mathcal{D}_s| = 20\%|\mathcal{D}|$. We mined top-10 rules on each $\mathcal{D}_s$ using PTopKDivMiner and PTopk-Miner and compute the corresponding F1. The p-values are 2.7×10^{-6} and 1.82×10^{-7} on Hospital and Airport, respectively, justifying the statistical difference between the two algorithms.

Varying k (P/R). To illustrate better, we report Recall of real error detection, by varying k from 1 to 30 in Figure 1(m). PTopKDivMiner consistently has higher Recall than all the baselines. Given a larger k, all methods are able to detect more real errors. However, since Precision is mainly impacted by the accuracy of REEs, rather than the number of REEs, it is insensitive in most cases (not shown).

Usefulness. We compared the relevance labels for PTopKDivMiner, PTopk-Miner and Random. Consistent with the previous results, on average, 78% REEs of PTopKDivMiner are labeled as relevant, as opposed to 64% (resp. 42%) by PTopk-Miner (resp. Random).

Exp-3: Effectiveness of diversity measures. Fixing the relevance measure as $\mathcal{M}_{\text{rel}}$, we tested the accuracy of each (and the combination) of our diversity measures using the metrics above.

Error coverage. Fixing $\eta = 0.85$ and $k = 5$, we tested PTopk-Miner, Random, PTopKDivMiner, PTopKDivMiner$_{\text{NoDiv}}$, and the four variants of PTopKDivMiner with a single diversity measure. As shown in Figure 1(n), the REEs of PTopKDivMiner catch 76.5% of errors of those detected by the entire set Σ_{all}, *i.e.,* its ECratio is 76.5%. In contrast, without diversity measures, the ECratio of PTopk-Miner, Random and PTopKDivMiner$_{\text{NoDiv}}$ is lower. In other words, our diversity measures reduce the redundancy of rules and catch diverse errors. Note that the ECratio of PTopk-Miner is the lowest (close to 0%). This is because (a) its REEs are "homogeneous", (b) its relevance model is trained by direct rule comparison, rather than based on unknown errors detected, and worse still, (c) since it only considers relevance, its REEs have high confidence, leading to few rule violations.

Method	Rank	Rule (note that we classify the impacts of authors into classes (*e.g.,* "Low", "Medium", "High" and "Very High"), based on the widely used measures h_index and p_index.)
PTopKDivMiner	φ_1	author(t_0) $\wedge$ author(t_1) $\wedge$ t_0.#citations = t_1.#citations $\wedge$ t_1.acamedic_impact = "High" $\rightarrow$ t_0.acamedic_impact = "High"
	φ_2	author(t_0) $\wedge$ author(t_1) $\wedge$ t_0.p_index = t_1.p_index $\wedge$ t_1.#published_papers $\leq$ 50 $\wedge$ t_0.#citations > 100 $\rightarrow$ t_0.acamedic_impact = "Medium"
	φ_3	author(t_0) $\wedge$ author(t_1) $\wedge$ author2paper(t_2) $\wedge$ author2paper(t_3) $\wedge$ paper(t_4) $\wedge$ paper(t_5) $\wedge$ t_0.author_id = t_2.author_id $\wedge$ t_2.paper_id = t_4.paper_id $\wedge$ t_1.author_id = t_3.author_id $\wedge$ t_3.paper_id = t_5.paper_id $\wedge$ $\mathcal{M}_{\text{sim}}$(t_4.venue, t_5.venue) $\wedge$ t_0.#published_papers = t_1.#published_papers $\wedge$ t_1.p_index = 0 $\rightarrow$ t_0.acamedic_impact = "Low"
PTopk-Miner	φ_1'	author(t_0) $\wedge$ author(t_1) $\wedge$ t_0.author_id = t_1.author_id $\wedge$ t_1.acamedic_impact = "Medium" $\rightarrow$ t_0.acamedic_impact = "Medium"
	φ_2'	author(t_0) $\wedge$ author(t_1) $\wedge$ t_0.p_index = t_1.p_index $\wedge$ t_1.acamedic_impact = "Medium" $\rightarrow$ t_0.acamedic_impact = "Medium"
	φ_3'	author(t_0) $\wedge$ author(t_1) $\wedge$ t_0.#citations = t_1.#citations $\wedge$ t_1.acamedic_impact = "Medium" $\rightarrow$ t_0.acamedic_impact = "Medium"

Table 5. Top-3 REEs by PTopKDivMiner and PTopk-Miner ($\sigma \geq 5 \times 10^{-6} \cdot |\mathcal{D}|^2$ and $\eta \geq 0.7$)

All diversity measures have comparable ECratio, except tuple coverage, whose ECratio is higher. This is because its goal is to "cover as many different tuple as possible", which is in favor of ECratio.

*Varying k (*ECratio*).* Fixing $\eta = 0.85$, we varied k from 1 to 30 and studied its impact on ECratio in Figure 1(o). For better visualization, we only plotted the best variant PTopKDivMiner$_{\text{TC}}$ and omitted others. As expected, with more rules, all algorithms detect more errors. This said, PTopKDivMiner performs the best, *e.g.,* its ECratio is 77.5% with 10 rules and beats the best baseline by 21.32% on average. Note that although PTopKDivMiner$_{\text{TC}}$ initially covers more errors (which it is designed for), PTopKDivMiner has higher ECratio when $k \geq 15$, by combining all diversity measures.

Error dissimilarity. As shown in Figure 1(p), the errors detected by PTopKDivMiner are dissimilar to each other, *e.g.,* its JD on Hospital is as high as 0.94, indicating that the rules it returns are not homogeneous and thus can be considered as non-redundant to practitioners. In contrast, the errors detected by other methods are more or less similar, yielding small JD. PTopk-Miner has the lowest JD.

*Varying λ (*JD*).* Varying the trade-off parameter λ from 0.1 to 0.025, we evaluated its impact on error dissimilarity in Figure 1(q). Here smaller λ means we focus more on diversity than relevance in our bi-criteria objective. As a result, the rules returned are able to detect more dissimilar errors in principle, *e.g.,* when λ is from 0.1 to 0.025, the JD of PTopKDivMiner increases from 0.47 to 0.685 on Airport.

Usefulness test. We compared the number of diversity labels received by PTopKDivMiner, PTopk-Miner and Random. On average, 86% pairs of REEs in PTopKDivMiner are labeled as dissimilar, as opposed to 68.5% and 71% by PTopk-Miner and Random, respectively. This further justifies the rules identified by PTopKDivMiner are diverse and they can cover/characterize different cases of errors.

Exp-4: Effectiveness of top-k relevant and diversified discovery. Finally, we justified the effectiveness of PTopKDivMiner.

Overall score. In Figure 1(r), we summed up the relevance and diversity scores as the overall score $F(\Sigma)$ of PTopKDivMiner, PTopk-Miner, Random, PTopKDivMiner$_{\text{NoDiv}}$ and PTopKDivMiner$_{\text{NoRel}}$. The rules from PTopKDivMiner achieve the best overall score; on average, it is 0.32, 0.34, 0.16 and 0.14 higher than the four baselines, respectively, because we optimize both diversity and relevance.

Optimality test. To verify the effectiveness of PTopKDivMiner, we compared its $F(\Sigma)$ with the (sub-)optimal one. We sampled 5K tuples from $\mathcal{D}$ and run REEFinder ($\sigma = 10^{-4}|\mathcal{D}|^2$ and $\eta = 0.75$), to get the set Σ_{all}. Given $k = 5$, it is too expansive to compute the set Σ_{opt} of k REEs with the optimal $F(\Sigma_{\text{opt}})$ (Theorem 2). Thus, we estimated it by a swapping strategy. Specifically, we sampled 100 subsets Σ_s of rules from Σ_{all} with $|\Sigma_s| = k$; for each Σ_s, we iteratively swapped a rule in Σ_s with a rule in Σ_{all} to best improve $F(\Sigma_s)$, until it reaches a local optimum, and Σ_{opt} is the one with the largest $F(\Sigma_s)$.

Figure 1(s) shows that the set Σ returned by PTopKDivMiner is as good as Σ_{opt}. Although Σ and Σ_{opt} are not the same in all the datasets, their overall scores do not differ much, *e.g.,* at most 0.0368.

Varying λ (relevance/diversity scores). Varying λ from 0.1 to 0.025, we tested its impact on the evaluation scores of PTopKDivMiner in Figure 1(t). To illustrate better, we plotted the relevance and diversity scores separately. When λ gets smaller, we focus more on diversity than relevance and thus, the relevance (resp. diversity) score of the resulting set of rules gets smaller (resp. larger), as expected.

Case study. We showcase the rules from PTopKDivMiner and PTopk-Miner in Table 5 in DBLP, which has 3 relations author, paper and author2paper (which connects the first two). It includes various metrics that reflect one's academic achievements, *e.g.,* h_index and p_index [55]. We mined rules with $k = 3$ to classify the impact of authors into different classes (*e.g.,* "Low", "Medium", "High" and "Very High"), where $\mathcal{M}_{\text{sim}}$ is an ML model for checking the similarity between publication venues.

(1) (Syntactic) The top-3 rules from PTopk-Miner are simple, with fewer and less diverse attributes/predicates than PTopKDivMiner. Worse still, all three rules focused on medium-impact authors.

(2) (Semantic) Some rules from PTopk-Miner describe trivial facts, *e.g.,* φ_1' says that if t_0 and t_1 are identified by the unique id and t_1 has medium impact, so does t_0, Worse still, φ_2' and φ_3' only differ in how they compare the authors' impact (*i.e.,* p_index vs. #citations) and thus, they are "homogeneous" and to some extent, "redundant".

In contrast, the top-3 rules from PTopKDivMiner not only include easy-to-understand rules like φ_1, which uses the impact of another researcher with same citation counts, to deduce one's academic impact, but also more complicated ones, *e.g.,* φ_3 that makes the decision by considering the p_index of another researcher who (a) has published the same number of papers and (b) has published some papers at similar venues. Better still, each rule uses a distinct set of attributes/predicates and covers a different impactful level.

Summary. We find the following. (1) Discovery of top-k relevant and diversified rules is efficient. For all $k \leq 40$, PTopKDivMiner is on average 62.4X faster than mining the entire set Σ_{all} of rules. It is feasible in practice: it takes less than 818s to mine top-10 REEs from NCVoter with 1.68M tuples when $n = 20$. (2) It is parallelly scalable: on average, it is 3.05X faster when the number n of machines varies from 4 to 20. (3) It scales well with parameters σ, η and k. (4) Our pruning strategies are effective, *e.g.,* it reduces the runtime of PTopKDivMiner by 51.66X when $k = 1$. (5) Our relevance model $\mathcal{M}_{\text{rel}}$ is accurate and its data-dependent embedding strategy improves the accuracy by 2.35%. (6) The rules returned by PTopKDivMiner are relevant (*e.g.,* its average F1 of real error detection is 0.74 when k = 10, 39.62% higher than PTopk-Miner) and diverse (*e.g.,* 10 rules can catch 77.5% of errors detected by Σ_{all}).

7 Conclusion

The work is novel in the following: (1) a discovery paradigm for top-k relevant and diversified rules; (2) a relevance model and four diversity measures for rule discovery; (3) the complexity and approximation hardness of the discovery problem; and (4) the first algorithm and pruning techniques for the problem, with parallel scalability and approximation bounds under certain conditions. We have empirically verified that the method is promising in practice.

One topic for future work is to further simplify and speed up the process of learning users' prior knowledge for relevance. Another topic is to develop an algorithm to incrementally discover rules.

Acknowledgments

This work is supported by China NSFC 62202313 and Guangdong Basic and Applied Basic Research Foundation 2022A1515010120.

References

[1] 2022. EJML library. http://ejml.org/wiki/index.php?title=Main_Page.
[2] 2024. Apple Corps Ltd. https://en.wikipedia.org/wiki/Apple_Corps.
[3] 2024. Apple Inc. https://www.apple.com/.
[4] 2024. China Organization Code - An Introduction. https://www.chinacheckup.com/blog/china-organization-code.
[5] 2024. Full version. https://drive.google.com/drive/folders/1iEZAzt6xj8K-A-8oK5XMkLBLn-Vtb_SV?usp=sharing.
[6] 2024. How To Find Your Company's Unified Social Credit Code in China. https://osome.com/hk/blog/how-to-find-your-companys-uscc-in-china/.
[7] 2024. Rock. http://www.grandhoo.com/en.
[8] Ziawasch Abedjan, Patrick Schulze, and Felix Naumann. 2014. DFD: Efficient functional dependency discovery. In *CIKM*. 949–958.
[9] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases*. Addison-Wesley.
[10] Efrat Abramovitz, Daniel Deutch, and Amir Gilad. 2018. Interactive inference of SPARQL queries using provenance. In *ICDE*. IEEE, 581–592.
[11] Rakesh Agrawal and Ramakrishnan Srikant. 1994. Fast Algorithms for Mining Association Rules in Large Databases. In *VLDB*.
[12] Elaine Angelino, Nicholas Larus-Stone, Daniel Alabi, Margo I. Seltzer, and Cynthia Rudin. 2017. Learning Certifiably Optimal Rule Lists for Categorical Data. *J. Mach. Learn. Res.* 18 (2017), 234:1–234:78.
[13] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. 1999. Consistent Query Answers in Inconsistent Databases. In *PODS*. 68–79.
[14] Xianchun Bao, Zian Bao, Qingsong Duan, Wenfei Fan, Hui Lei, Daji Li, Wei Lin, Peng Liu, Zhicong Lv, Mingliang Ouyang, Jiale Peng, Jing Zhang, Runxiao Zhao, Shuai Tang, Shuping Zhou, Yaoshu Wang, Qiyuan Wei, Min Xie, Jing Zhang, Xin Zhang, Runxiao Zhao, and Shuping Zhou. 2024. Rock: Cleaning Data by Embedding ML in Logic Rules. In *SIGMOD (industrial track)*.
[15] Arthur G Bedeian and Kevin W Mossholder. 2000. On the use of the coefficient of variation as a measure of diversity. *Organizational Research Methods* 3, 3 (2000), 285–297.
[16] Indrajit Bhattacharya and Lise Getoor. 2007. Collective entity resolution in relational data. *TKDD* (2007).
[17] Tobias Bleifuß, Sebastian Kruse, and Felix Naumann. 2017. Efficient Denial Constraint Discovery with Hydra. *PVLDB* 11, 3 (2017), 311–323.
[18] Allan Borodin, Aadhar Jain, Hyun Chul Lee, and Yuli Ye. 2017. Max-sum diversification, monotone submodular functions, and dynamic updates. *ACM Transactions on Algorithms (TALG)* 13, 3 (2017), 1–25.
[19] Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. 1984. *Classification and Regression Trees*. Wadsworth.
[20] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. 1993. Signature verification using a "Siamese" time delay neural network. In *Advances in neural information processing systems 6*.
[21] Jaime Carbonell and Jade Goldstein. 1998. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *SIGIR*. ACM, 335–336.
[22] Barun Chandra and Magnús M Halldórsson. 2001. Approximation algorithms for dispersion problems. *Journal of algorithms* 38, 2 (2001), 438–465.
[23] Chaofan Chen and Cynthia Rudin. 2018. An Optimization Approach to Learning Falling Rule Lists. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, Vol. 84. PMLR, 604–612.
[24] Kewei Cheng, Jiahao Liu, Wei Wang, and Yizhou Sun. 2022. RLogic: Recursive Logical Rule Learning from Knowledge Graphs. In *KDD*. ACM, 179–189.
[25] 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. https://doi.org/10.1145/3418896
[26] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. 2013. Discovering Denial Constraints. *PVLDB* 6, 13 (2013), 1498–1509.
[27] William W. Cohen. 1995. Fast Effective Rule Induction. In *International Conference on Machine Learning*. Morgan Kaufmann, 115–123.
[28] Anirban Dasgupta, Ravi Kumar, and Sujith Ravi. 2013. Summarization Through Submodularity and Dispersion. In *ACL*. ACL, 1014–1022.
[29] Ting Deng and Wenfei Fan. 2014. On the Complexity of Query Result Diversification. *ACM Trans. Database Syst.* 39, 2 (2014), 15:1–15:46.
[30] 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.
[31] Wenfei Fan. 2022. Big Graphs: Challenges and Opportunities. *PVLDB* 15, 12 (2022), 3782–3797.
[32] Wenfei Fan, Hong Gao, Xibei Jia, Jianzhong Li, and Shuai Ma. 2011. Dynamic constraints for record matching. *VLDB J.* 20, 4 (2011), 495–520.

[33] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2008. Conditional Functional Dependencies for Capturing Data Inconsistencies. *ACM Trans. Database Syst.* 33, 1 (2008), 25:1–25:49.
[34] Wenfei Fan, Floris Geerts, Jianzhong Li, and Ming Xiong. 2011. Discovering conditional functional dependencies. *TKDE* 23, 5 (2011), 683–698.
[35] Wenfei Fan, Ziyan Han, Yaoshu Wang, and Min Xie. 2022. Parallel Rule Discovery from Large Datasets by Sampling. In *SIGMOD*. ACM, 384–398.
[36] Wenfei Fan, Ziyan Han, Yaoshu Wang, and Min Xie. 2023. Discovering Top-k Rules using Subjective and Objective Criteria. *Proc. ACM Manag. Data* 1, 1 (2023), 70:1–70:29.
[37] Wenfei Fan, Ping Lu, and Chao Tian. 2020. Unifying logic rules and machine learning for entity enhancing. *Sci. China Inf. Sci.* 63, 7 (2020).
[38] Wenfei Fan, Chao Tian, Yanghao Wang, and Qiang Yin. 2021. Parallel Discrepancy Detection and Incremental Detection. *PVLDB* 14, 8 (2021), 1351–1364.
[39] Wenfei Fan, Xin Wang, and Yinghui Wu. 2013. Diversified Top-k Graph Pattern Matching. *PVLDB* 6, 13 (2013), 1510–1521.
[40] Wenfei Fan, Xin Wang, Yinghui Wu, and Jingbo Xu. 2015. Association Rules with Graph Patterns. *PVLDB* 8, 12 (2015), 1502–1513.
[41] Wenfei Fan, Yinghui Wu, and Jingbo Xu. 2016. Functional dependencies for graphs. In *SIGMOD*. 1843–1857.
[42] Uriel Feige. 1998. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)* 45, 4 (1998), 634–652.
[43] Peter A Flach and Iztok Savnik. 1999. Database dependency discovery: A machine learning approach. *AI communications* 12, 3 (1999), 139–160.
[44] Philippe Fournier-Viger and Vincent S Tseng. 2012. Mining top-k non-redundant association rules. In *Foundations of Intelligent Systems (ISMIS)*. Springer, 31–40.
[45] Piero Fraternali, Davide Martinenghi, and Marco Tagliasacchi. 2012. Top-k bounded diversification. In *SIGMOD*. ACM, 421–432.
[46] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. 2007. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics* 9, 3 (2007), 432–441.
[47] Johannes Fürnkranz and Gerhard Widmer. 1994. Incremental Reduced Error Pruning. In *International Conference on Machine Learning*. Morgan Kaufmann, 70–77.
[48] Michael Garey and David Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.
[49] Chang Ge, Ihab F. Ilyas, and Florian Kerschbaum. 2019. Secure Multi-Party Functional Dependency Discovery. *PVLDB* 13, 2 (2019), 184–196.
[50] Liqiang Geng and Howard J Hamilton. 2006. Interestingness measures for data mining: A survey. *ACM Computing Surveys (CSUR)* 38, 3 (2006), 9–es.
[51] Lukasz Golab, Howard Karloff, Flip Korn, Divesh Srivastava, and Bei Yu. 2008. On generating near-optimal tableaux for conditional functional dependencies. *PVLDB* 1, 1 (2008), 376–390.
[52] Sreenivas Gollapudi and Aneesh Sharma. 2009. An axiomatic approach for result diversification. In *WWW*.
[53] Johan Hastad. 1996. Clique is hard to approximate within n/sup 1-/spl epsiv. In *FOCS*. IEEE, 627–636.
[54] Robert J Hilderman and Howard J Hamilton. 2013. *Knowledge discovery and measures of interest*. Vol. 638. Springer Science & Business Media.
[55] Adrian Horzyk. 2014. p-index–A Fair Alternative to h-index. *Department of Automatics and Biomedical Engineering. Poland* (2014).
[56] Xiyang Hu, Cynthia Rudin, and Margo I. Seltzer. 2019. Optimal Sparse Decision Trees. In *NeurIPS*. 7265–7273.
[57] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. 1999. TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies. *Comput. J.* (1999).
[58] Zhengbao Jiang, Ji-Rong Wen, Zhicheng Dou, Wayne Xin Zhao, Jian-Yun Nie, and Ming Yue. 2017. Learning to diversify search results via subtopic attention. In *SIGIR*. 545–554.
[59] Jyrki Kivinen and Heikki Mannila. 1995. Approximate inference of functional dependencies from relations. *Theoretical Computer Science* 149, 1 (1995), 129–149.
[60] loannis Koumarelas, Thorsten Papenbrock, and Felix Naumann. 2020. MDedup: Duplicate detection with matching dependencies. *PVLDB* 13, 5 (2020), 712–725.
[61] Sebastian Kruse and Felix Naumann. 2018. Efficient discovery of approximate dependencies. *PVLDB* 11, 7 (2018), 759–772.
[62] Clyde P. Kruskal, Larry Rudolph, and Marc Snir. 1990. A Complexity Theory of Efficient Parallel Algorithms. *Theor. Comput. Sci.* 71, 1 (1990), 95–132.
[63] Himabindu Lakkaraju, Stephen H Bach, and Jure Leskovec. 2016. Interpretable decision sets: A joint framework for description and prediction. In *SIGKDD*. ACM, 1675–1684.

[64] J Kevin Lanctot, Ming Li, Bin Ma, Shaojiu Wang, and Louxin Zhang. 2003. Distinguishing string selection problems. *Information and Computation* 185, 1 (2003), 41–55.

[65] Marie Le Guilly, Jean-Marc Petit, and Vasile-Marian Scuturici. 2020. Evaluating classification feasibility using functional dependencies. *Transactions on Large-Scale Data-and Knowledge-Centered Systems XLIV: Special Issue on Data Management–Principles, Technologies, and Applications* (2020), 132–159.

[66] Adam Lerer, Ledell Wu, Jiajun Shen, Timothée Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. 2019. Pytorch-BigGraph: A Large Scale Graph Embedding System. In *MLSys*.

[67] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep Entity Matching with Pre-Trained Language Models. *PVLDB* 14, 1 (2020), 50–60.

[68] Jimmy Lin, Chudi Zhong, Diane Hu, Cynthia Rudin, and Margo I. Seltzer. 2020. Generalized and Scalable Optimal Sparse Decision Trees. In *International Conference on Machine Learning (ICML)*, Vol. 119. PMLR, 6150–6160.

[69] Weiwen Liu, Yunjia Xi, Jiarui Qin, Xinyi Dai, Ruiming Tang, Shuai Li, Weinan Zhang, and Rui Zhang. 2023. Personalized Diversification for Neural Re-ranking in Recommendation. In *ICDE*. IEEE, 802–815.

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

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

[72] Mohammad Mahdavi and Ziawasch Abedjan. 2020. Baran: Effective error correction via a unified context representation and transfer learning. *PVLDB* 13, 12 (2020), 1948–1961.

[73] Panagiotis Mandros, Mario Boley, and Jilles Vreeken. 2017. Discovering reliable approximate functional dependencies. In *SIGKDD*. 355–363.

[74] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. 1994. Efficient Algorithms for Discovering Association Rules. In *Knowledge Discovery in Databases: AAAI*. AAAI Press, 181–192.

[75] Michel Minoux. 2005. Accelerated greedy algorithms for maximizing submodular set functions. In *IFIP Conference on Optimization Techniques*. Springer, 234–243.

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

[77] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. 1978. An analysis of approximations for maximizing submodular set functions—I. *Mathematical programming* 14 (1978), 265–294.

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

[79] Donald B Owen. 1965. The power of Student's t-test. *J. Amer. Statist. Assoc.* 60, 309 (1965), 320–333.

[80] 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.

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

[82] Marcel Parciak, Sebastiaan Weytjens, Niel Hens, Frank Neven, Liesbet M Peeters, and Stijn Vansummeren. 2023. Measuring Approximate Functional Dependencies: a Comparative Study. *arXiv preprint arXiv:2312.06296* (2023).

[83] 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.

[84] Chao Qian, Dan-Xuan Liu, and Zhi-Hua Zhou. 2022. Result diversification by multi-objective evolutionary algorithms with theoretical guarantees. *Artificial Intelligence* 309 (2022), 103737.

[85] Lu Qin, Jeffrey Xu Yu, and Lijun Chang. 2012. Diversifying top-κ results. *PVLDB* (2012).

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

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

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

[89] Sekharipuram S Ravi, Daniel J Rosenkrantz, and Giri Kumar Tayi. 1994. Heuristic and special case algorithms for dispersion problems. *Operations research* 42, 2 (1994), 299–310.

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

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

[92] Rock. 2024. Entity Deduplication in a Major Domestic Research Institution. https://www.grandhoo.com/en/rock/customer-case/entity-reduction/.

[93] Rock. 2024. Keyword Search in an Open Data Platform. https://www.grandhoo.com/en/rock/customer-case/field-search/.

[94] Rock. 2024. Multi-source Entity Resolution in a Commercial Bank. https://www.grandhoo.com/en/rock/customer-case/bank-entity/.
[95] Rock. 2024. Regulatory Reporting in a Leading Bank in Shenzhen. https://www.grandhoo.com/en/rock/customer-case/bank-supervision/.
[96] Sudeepa Roy and Dan Suciu. 2014. A formal approach to finding explanations for database queries. In *SIGMOD*. ACM, 1579–1590.
[97] Cynthia Rudin, Benjamin Letham, and David Madigan. 2013. Learning theory analysis for association rules and sequential event prediction. *J. Mach. Learn. Res.* 14, 1 (2013), 3441–3492.
[98] Philipp Schirmer, Thorsten Papenbrock, Ioannis Koumarelas, and Felix Naumann. 2020. Efficient Discovery of Matching Dependencies. *RODS* 45, 3 (2020), 1–33.
[99] 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*.
[100] 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.
[101] Shaoxu Song and Lei Chen. 2009. Discovering matching dependencies. In *CIKM*.
[102] Ramakrishnan Srikant and Rakesh Agrawal. 1996. Mining Sequential Patterns: Generalizations and Performance Improvements. In *EDBT*. Springer, 3–17.
[103] Zheng Tan, Hang Yu, Wei Wei, and Jinglei Liu. 2020. Top-K interesting preference rules mining based on MaxClique. *Expert Systems with Applications* 143 (2020), 113043.
[104] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. ArnetMiner: Extraction and Mining of Academic Social Networks. In *KDD*. 990–998.
[105] Marcos R. Vieira, Humberto Luiz Razente, Maria Camila Nardini Barioni, Marios Hadjieleftheriou, Divesh Srivastava, Caetano Traina Jr., and Vassilis J. Tsotras. 2011. On query result diversification. In *ICDE*.
[106] Tong Wang, Cynthia Rudin, Finale Doshi-Velez, Yimin Liu, Erica Klampfl, and Perry MacNeille. 2017. A Bayesian Framework for Learning Rule Sets for Interpretable Classification. *J. Mach. Learn. Res.* 18 (2017), 70:1–70:37.
[107] G. I. Webb and S. Zhang. 2005. k-Optimal Rule Discovery. *Data Mining and Knowledge Discovery* 10, 1 (2005), 39–79.
[108] Eugene Wu and Samuel Madden. 2013. Scorpion: Explaining Away Outliers in Aggregate Queries. *PVLDB* 6, 8 (2013).
[109] 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*.
[110] Dong Xin, Hong Cheng, Xifeng Yan, and Jiawei Han. 2006. Extracting redundancy-aware top-k patterns. In *SIGKDD*. 444–453.
[111] Zhengwei Yang, Ada Wai-Chee Fu, and Ruifeng Liu. 2016. Diversified top-k subgraph querying in a large graph. In *SIGMOD*. 1167–1182.
[112] Hong Yao, Howard J. Hamilton, and Cory J. Butz. 2002. FD_Mine: Discovering functional dependencies in a database using equivalences. In *IEEE ICDM*. 1–15.
[113] Jinsung Yoon, James Jordon, and Mihaela van der Schaar. 2018. GAIN: Missing Data Imputation using Generative Adversarial Nets. In *ICML (Proceedings of Machine Learning Research)*, Vol. 80. PMLR, 5675–5684.
[114] Long Yuan, Lu Qin, Xuemin Lin, Lijun Chang, and Wenjie Zhang. 2016. Diversified top-k clique search. *VLDBJ* 25, 2 (2016), 171–196.
[115] Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. 1997. New Algorithms for Fast Discovery of Association Rules. In *KDD*. AAAI Press, 283–286.
[116] Guangyi Zhang and Aristides Gionis. 2020. Diverse rule sets. In *SIGKDD*. ACM, 1532–1541.
[117] Yunjia Zhang, Zhihan Guo, and Theodoros Rekatsinas. 2020. A Statistical Perspective on Discovering Functional Dependencies in Noisy Data. In *SIGMOD*. 861–876.

Received January 2024; revised April 2024; accepted May 2024