Discovering Top-k Rules using Subjective and Objective Criteria

Paper id: XXX

ABSTRACT

This paper studies two questions about rule discovery. Can we characterize the usefulness of rules using quantitative criteria? How can we discover rules using those criteria? As a testbed, we consider *entity enhancing rules* (REEs), which subsume common association rules and data quality rules as special cases. We characterize REEs using a bi-criteria model, with both objective measures such as support and confidence, and subjective measures for the user's needs; we learn the subjective measure and the weight vectors via active learning. Based on the bi-criteria model, we develop a top-k algorithm to discover top-ranked REEs, and an any-time algorithm for successive discovery via lazy evaluation. We parallelize these algorithms such that they guarantee to reduce runtime when more processors are used. Using real-life and synthetic datasets, we show that the algorithms are able to find top-ranked rules and speed up conventional rule-discovery methods by 86 times on average.

1 INTRODUCTION

Rules play a critical role in many aspects of data management, *e.g.,* association rules reveal hidden regularities among entities, entity resolution (ER) rules identify tuples that refer to the same entity, and conflict resolution (CR) rules resolve conflicts pertaining to the entities. Recently, rule-based methods find new applications in *drug repurposing* to treat new diseases with known drugs, and *adverse drug reaction (ADR) prediction* to identify undesirable effects [100]. Drug discovery is a costly and time-consuming process, starting from target selection and validation, through preclinical screening, to clinical trials [36]. On average, the development of a new drug takes 15 years [23] and costs 800 million dollars [6], accompanied by a high risk of failure (>90% [11]). To shorten the discovery cycle, reduce the cost and increase the success rate, computational methods have been explored for identifying drug-disease associations (DDA) and drug-drug interaction (DDI). There has also been increasing need for ER and CR to reduce false relations caused by noise, and for explanations to justify discovered DDA and DDI [100].

To make practical use of rules, it is a must to be able to discover rules from real-life data. However, a major problem in practice is that rule discovery often yields *excessive* rules, *e.g.,* on a small dataset with 27 attributes and 368 tuples, 128,726 functional dependencies (FDs) are found [72]. Practitioners 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 needs. This staggering cost hampers the applicability of rule-based methods.

Typically, domain experts have accumulated a collection of "rules" from their practice, *e.g.,* they already know some well-understood uses of the drugs. These rules might be mined using conventional measures, *e.g.,* support and confidence for how often the rules can be applied and how strong the associations between their preconditions and consequences are. However, these "universal" objective measures often do not suffice in practice. What the domain experts want from rule discovery are rules that are "surprising" or "novel" to them, *e.g.,* rules that help them identify new uses of drugs, to complement those they have already got. For an abundant of candidate rules returned by traditional discovery methods, the experts often find them not equally potent for therapeutic intervention. They only want the most promising ones and prioritize them for the next phases, *e.g.,* for costly clinical trials, since they cannot afford to try them all. They want more rules to be retrieved only if they find the selected ones unsatisfactory. Our fraud-detection users also want only rules for new fraud patterns.

This gives rise to several questions. How can we discover top-ranked "novel" and "surprising" rules for the costly trials, to complement those that the domain experts have already known? If the selected candidates are not satisfactory, can we find more rules efficiently, without starting from scratch? How can we parallelize the process and scale with the increasing biomedical data? Can we interpret discovered DDA and DDI, and cope with noise, which are critical but are still open in pharmaceutical scenarios [100]?

Contributions & organizations. This paper tackles these issues, exploring a new approach to discovering top-ranked rules. As proof of concept of our proposed methods, we consider entity enhancing rules (REEs) [33], which were originally designed for ER and CR, and recently find new applications in association analysis and prediction interpretability. REEs (a) embed ML classifiers in logic rules, (b) unify ER, CR and association analysis in a uniform framework, (c) are collectively defined across multiple tables, and (d) subsume association rules [77], matching dependencies (MDs) [10, 12, 27], denial constraints (DCs) [9] and conditional functional dependencies (CFDs) [28, 30] as special cases. REEs are used by Rock, an industrial system for drug discovery. We will review REEs in Section 2.

(1) Bi-criteria model (Section 3). We propose a *bi-criteria* model to characterize rules, in terms of (a) conventional objective measures, and (b) new subjective measures to fit users' needs. The two parts bear various weights to serve different users. Experienced users may assign a higher weight to the subjective measure in favor of surprising rules, while novice users may opt to prioritize objective measures to find rules with high support and confidence. Since it is unrealistic to ask users to specify the weights explicitly, we propose an active learning method to learn the user's need. In addition, we show that the conventional notion of support does not fit collective rules such as REEs, *i.e.,* it is no longer anti-monotonic; in light of this, we revise the notion for REEs and show its anti-monotonicity.

(2) Discovering top-ranked rules. We develop a set of techniques:

- Based on the bi-criteria model, we develop a top-k discovery algorithm (Section 4), which reduces excessive rules. The algorithm learns a score bound and terminates early as soon as it finds top-k rules. It is far less costly than conventional rule discovery algorithms, which first find all the rules that hold on a dataset, and then sort the rules and return top-k ones.
- Users often want to continue to find the next top-k rules if they are not satisfied with the current ones; this is analogous to how we use search engines. In response to this we provide an anytime-algorithm to find the next top-k rules if needed, via

lazy evaluation (Section 5). By repeatedly running the algorithm, users can also find all the rules that hold on a dataset in order.
- To scale with large data, we parallelize the top-k algorithm and the anytime algorithm across a cluster of machines (Section 6). We show that the algorithms are parallelly scalable [55], *i.e.,* they guarantee to reduce the runtime when more machines are used. Hence the algorithms are able to efficiently discover rules when the data grows big, by adding more computing resources.

(3) Experimental study (Section 7). Using real-life and synthetic datasets, we empirically find the following. On average, (a) top-k REEs discovery speeds up conventional methods by 86 times. It takes 183s on NCVoter with 1,681,617 tuples for $k = 10$ with 20 machines, versus 12,003s by traditional methods. (b) The lazy evaluation strategy makes the anytime algorithm 52.8 times faster than the top-k one when users want the 4th top-10 REEs. (c) Our bi-criteria model is 15% more accurate than the state-of-the-art language models, and its subjective measure improves the accuracy from 0.525 to 0.76. (d) The algorithms are parallelly scalable; they are 3.12 times faster using 20 machines instead of 4.

To the best of our knowledge, this work makes the first effort to employ a bi-criteria model for rule discovery. It also provides the first top-k and anytime algorithms, with the parallel scalability.

Related work. The related work is categorized as follows.

Rule discovery. A number of rule discovery algorithms have been developed for ER, CR and association analysis, classified as follows. (1) *Levelwise search.* TANE [50], FUN [71], FD_mine [97] discover FDs based on a lattice structure, which is latter extended by Depmine [66], HyFD [73] DynFD [85], and SMFD [41]. Levelwise methods for CFDs and MDs include CTANE [29], tableau generation [44] and [88]. [31] adopts sampling to discover REEs in large datasets. Apriori [7] and its variants (*e.g.,* DIC [15] and GSP [89]) employ breath-first search to mine association rules or frequent itemsets. (2) *Depth-first search.* DFD [4] and FastFDs [95] adopt the depth-first search in the lattice to mine FDs. For CFDs and DCs, depth-first search approaches also apply, *e.g.,* FastCFDs [29], FastDC [20], Hydra [14], DCFinder [74] and ADCMiner [65]. Note that although DCs allow multiple relation atoms [20] by definition, all its discovery algorithms [14, 65, 74] are restricted to bi-variable DCs only. FP-growth [46] and Eclat [99] use depth-first search to mine association rules or frequent itemsets. (3) *Hybrid approaches.* HyMD [84] and MDedup [53] employ both levelwise and depth-first search to discover MDs. (4) *Learning-based approaches.* Inductive learning [35] and structure learning [101] have been utilized to find FDs. [87] and [52] adopt rule learning strategies to find MDs (see [19] for more ER solvers). Learned rules are also used for blocking and debugging, *e.g.,* Smurf [17]. Similar techniques also apply to association rule mining [81]. (5) *Top-k discovery.* Closer to this work are top-k algorithms for discovering association rules in relations [93] and graphs [34]. Mining of association rules and sequential patterns has also been studied in [39] and [91].

This work differs from the prior work as follows. (1) We propose a novel bi-criteria model that combines both objective and subjective measures, the first of the kind for ER, CR and association rules. (2) We use active learning and pairwise ranking to learn the ranking of rules, to capture the user's preference. (3) We extend our top-k algorithm to an anytime algorithm, for users to find the next top-k results when needed, without re-discovering starting from scratch. No existing work has considered anytime rule discovery.

Subjective and objective measures. The need for considering objective and subjective measures has long been recognized, to reflect users' universal and individualistic preference (see [42] for a survey). However, we are not aware of any rule discovery algorithms that take both measures into account. While MDedup [53] employs objective features for MDs and adopts the ML regression model and Gaussian Process to learn an MD score, it aims to discover MDs with high F-measures, not for users' surprisingness. While ML models [24, 86] were designed to learn the representations of rules, they do not consider how to learn subjective criteria for rules.

This work studies top-k rule discovery based on a bi-criteria model, and proposes a learning approach to quantifying subjective measures. It differs from [34, 93] in the use of different ranking criteria and thus different early-termination strategies.

ML models. ML models have been studied for ER, CR or association analysis. (1) *Models for ER and link prediction*, via logistic regression and SVM (see [43] for a survey), unsupervised learning, *e.g.,* ZeroER [94], and deep learning, *e.g.,* Ditto [61], BertER [57], DeepMatcher [70], DeepER [26], AutoEM [102], GraphER [58], MPM [38]. (2) *Models for CR*, including HoloClean [79], HoloDetect [48], Raha [67] and SLiMFast [80], for error detection, error correction and data fusion. (3) *Models for similarity checking*, by employing popular language models such as BERT-based models [22, 56, 64, 78], XLNet [96] and GPT [16, 76].

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

Parallel discovery. Parallel methods have been developed for rule discovery. [40, 59] employ parallelism to discover FDs without considering communication cost. [60] discovers local FDs in a distributed setting. [82] extends FastFD, minimizing communication cost. [83] proposes a distributed framework to discover FDs and DCs. SMFD [41] enforces privacy constraints in distributed FDs discovery. [68] introduces parallel FI-growth to mine frequent itemsets. DPF, STPF and DTPF [45] are tree-projection-based methods using partitioning. SPAMC [18] and Sequence-Growth [62] discover sequence patterns via MapReduce (see [39, 51] for surveys).

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

2 ENTITY ENHANCING RULES

We next review entity enhancing rules (REEs) defined in [33].

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

tid	cid (DrugBank)	name	type	weight	summary	formula	manufacturer
t_1	DB00915	Amantadine	Small Molecule	151.2487	A medication used to treat dyskinesia in Parkinson's patients...	$C_{10}H_{17}N$	Actavis totowa llc
t_2	DB00914	Phenformin	Small Molecule	205.2596	A biguanide hypoglycemic agent with actions and uses similar to ..	$C_{10}H_{15}N_5$	n/a
t_3	DB00937	Diethylpropion	Molecule	205.2961	An appetite suppressant for short term treatment of exogenous obesity...	$C_{13}H_{19}NO$	Sanofi aventis us llc
t_4	DB0091	Phenylethylbiguanide	Molecule	205.26	An agent belonging to the biguanide class of antidiabetics...	$C_{10}H_{15}N_5$	US Vitamin Corp.
t_5	DB000898	Ethanol	Small Molecule	46.0684	A clear, colorless liquid rapidly absorbed from the gastrointestinal tract...	C_2H_6O	Miles laboratories inc

Table 1: Example Drug relation D_1

tid	cid	uid (MeSH)	term	description	classification	established_date	revision_date
t_6	DB00915	D003972	Cognition Disorders	Disorders characterized by idsturbances in mental processes...	Mental	1969-01-01	2016-05-31
t_7	DB00914	D000749	Anemia	A disorder by the presence of ANEMIA, abnormally large red blood cells...	Hematologic	1991-01-01	2009-07-06
t_8	DB0091	D00074	Anemia	Disorder of anemia, large red blood cells...	Hematologic	1991-01-01	2009-070-06
t_9	DB00914	D001284	Atrophy	Decrease in the size of a cell, tissue, organ, or multiple organs...	Pathological	1966-01-01	1999-11-08
t_{10}	DB00937	D009765	Obesity	A status with BODY WEIGHT that is grossly above the standards...	Nutritional & Metabolic	1966-01-01	2021-07-07
t_{11}	DB00898	D000169	Acrodermatitis	Dermatitis of hands or feet so do not bother to...	Skin	1966-01-01	2015-06-23
t_{12}	DB00888	D004485	Eczema	A pruritic papulovesicular dermatitis occurring as a reaction to...	Tissues	1966-01-01	1992-05-08

Table 2: Example Disease relation D_2

is a collection $(D_1, \ldots, D_m)$, where D_i is a relation of R_i ($i \in [1, m]$).

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

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

where $\oplus$ is an operator in $\{=, \neq\}$. Following tuple relational calculus [5], (a) $R(t)$ is a *relation atom* over $\mathcal{R}$, where $R \in \mathcal{R}$, and t is a *tuple variable bounded by* $R(t)$; (b) $t.A$ denotes an attribute of t when t is bounded by $R(t)$ and A is an attribute in R; (c) $t.A \oplus c$ is *a constant predicate* when c is a value in the domain of A; and (d) $t.A \oplus s.B$ compares *compatible* attributes $t.A$ and $s.B$, *i.e.,* tuple t (resp. s) is bounded by $R(t)$ (resp. $R'(s)$), and $A \in R$ and $B \in R'$ have the same type. Moreover, (e) $\mathcal{M}(t[\bar{A}], s[\bar{B}])$ is *an ML predicate*, where $t[\bar{A}]$ and $s[\bar{B}]$ are vectors of pairwise compatible attributes.

Here $\mathcal{M}$ can be any existing ML model that returns a Boolean value, *e.g.,* $\mathcal{M}_{\text{reg}} \geq \delta$ for a regression model $\mathcal{M}_{\text{reg}}$ and a bound δ. In this paper, we consider $\mathcal{M}$ such as (1) NLP models, *e.g.,* Bert [22], for text classification; (2) ER models and link prediction models, *e.g.,* ditto [61] and DeepMatcher [70], to reveal "hidden" associations between tuples across relations, which are not connected by, *e.g.,* keys and foreign keys; and (3) models for data fusion, error detection and correction, *e.g.,* HoloClean [79] and HoloDetect [48].

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

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

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

Example 1: Consider an example from a (simplified) drug-disease database with self-explained schemas Drug (cid, name, type, weight, summary, formula, manufacturer) and Disease (cid, uid, term, description, classification, established_date, revision_date).

Below are example REEs over the database schema.

(1) $\varphi_1 : \text{Drug}(t_a) \wedge \text{Drug}(t_b) \wedge X \rightarrow \mathcal{M}_{\text{bio}}(t_a, t_b)$, where $\mathcal{M}_{\text{bio}}$ is a model for predicting the bioequivalence between two drugs, $X = \bigcap_{A_s \in \mathcal{T}} t_a.A_s = t_b.A_s$ and $\mathcal{T}$ denotes a designated set of biomedical features in Drug (not shown in the simplified schema), including dosage form, safety, strength, route of administration, performance characteristics, and intended use, etc. Here conditions in X interpret the prediction of $\mathcal{M}_{\text{bio}}$ in logic. Such interpretability is critical to pharmaceutical applications. This rule is consistent with the standard used by U.S. Food and Drug Administration (FDA) for identifying the equivalence of generic and brand-name drugs [63].

(2) φ_2: $\text{Drug}(t_a) \wedge \text{Disease}(s_a) \wedge \text{Disease}(s_b) \wedge t_a.\text{cid} = s_a.\text{cid} \wedge \mathcal{M}_{\text{therapy}}(s_a, s_b) \rightarrow t_a.\text{cid} = s_b.\text{cid}$. Here $\mathcal{M}_{\text{therapy}}$ is an ML model for checking the common therapies of two diseases. Intuitively, φ_2 tells that t_a is an efficacious treatment of s_b if the two diseases in s_a and s_b have common therapies (checked by $\mathcal{M}_{\text{therapy}}$), and t_a has been used to cure s_a. This rule expresses the pattern recognized in [25] and helps us discover the use of drug t_a for the disease in s_b since Disease has not recorded the fact "t_a can be used for s_b" yet.

(3) φ_3 : $\text{Drug}(t_a) \wedge \text{Disease}(s_a) \wedge \text{Disease}(s_b) \wedge t_a.\text{cid} = s_a.\text{cid} \wedge s_a.\text{term} = \text{"Atrophy"} \wedge \mathcal{M}_{\text{new}}(t_a, s_b) \wedge \mathcal{M}_{\text{therapy}}(s_a, s_b) \rightarrow s_b.\text{classification} = \text{"Pathological"}$, where $\mathcal{M}_{\text{therapy}}$ is as above, and $\mathcal{M}_{\text{new}}(t_a, s_b)$ is a link prediction model for telling whether a drug t_a can be used to cure s_b. Intuitively, φ_3 says that if (a) a drug t_a has been used for a disease s_a termed "Atrophy", which has common therapies as s_b (checked by $\mathcal{M}_{\text{therapy}}$), and (b) t_a is likely to be used for s_b, then s_b is classified as "Pathological". Note that here the fact "t_a cures s_b" is predicted by the link prediction model $\mathcal{M}_{\text{new}}$. This rules can be used to fix errors in the classification attribute. □

As shown above, REEs embed ML classifiers in logic rules, unifying ER, CR and association analysis. They may carry multiple tuple variables (*e.g.,* 3 variables in φ_2) for collective analysis [13]. We can deduce DDA/DDI, give explanations and fix errors using REEs.

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

We say that h *satisfies* a predicate p, written as $h \models p$, if the following are satisfied: (1) If p is a relation atom $R(t)$, $t \oplus c$ or $t.A \oplus s.B$, then $h \models p$ is interpreted as in tuple relational calculus following the standard semantics of first-order logic [5]. (2) If p is $\mathcal{M}(t[\bar{A}], s[\bar{B}])$, then $h \models p$ if $\mathcal{M}$ predicts true on $(h(t)[\bar{A}], h(s)[\bar{B}])$.

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

Example 2: Continuing with Example 1, assume that $\mathcal{D}$ consists of two relations D_1 and D_2 of schemas Drug and Disease, shown in Tables 1 and 2, respectively. Consider valuation h_2: $t_5 \mapsto t_a$, $t_{11} \mapsto s_a$ and $t_{12} \mapsto s_b$. This valuation satisfies φ_2, and helps us discover a new drug "Ethanol " for disease "Eczema". □

3 BI-CRITERIA RULE RANKING

In this section we study how to rank REEs. We first present the ranking criteria, including the objective and subjective measures (Section 3.1). We then propose our bi-criteria model (Section 3.2). Finally, we show how to learn the subjective model (Section 3.3).

3.1 Ranking Measures

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

3.1.1 Objective Measures. We start with objective measures.

Support. Support measures how frequently an REE can be applied. For collective rules across multiple tables such as REEs, the conventional notion of support has to be revised. To see this, we first define an order on REEs. Given two REEs $\varphi : X \rightarrow p_0$ and $\varphi' : X' \rightarrow p_0$ with the same consequence p_0, we say that φ has a *lower order* than φ', denoted by $\varphi \preceq \varphi'$, if $X \subseteq X'$. That is, φ is less restrictive than φ'.

Given a dataset $\mathcal{D}$ of schema $\mathcal{R}$, conventionally support for REEs $\varphi : X \rightarrow p_0$ over $\mathcal{R}$ is defined as the number of distinct valuations h of φ in $\mathcal{D}$ such that $h \models X$. This is the notion for rules on a *single* relation, *e.g.,* FDs, CFDs, etc. It satisfies the *anti-monotonicity* on single relations, *i.e.,* if $\varphi \preceq \varphi'$, then the support of φ is at least that of φ'. Unfortunately, for *collective* rules involving *multiple relations*, this definition does not work, as shown by the example below.

Example 3: Let X be $\text{Drug}(t_a) \wedge t_a.\text{name} = \text{"Phenformin"}$ and p_0 be $t_a.\text{formula} = \text{"C}_{10}\text{H}_{15}\text{N}_5\text{"}$. Consider two REEs φ and φ', $\varphi : X \rightarrow p_0$ and $\varphi' : X' \rightarrow p_0$, where $X' = X \wedge \text{Disease}(s_a) \wedge t_a.\text{cid} = s_a.\text{cid}$. Clearly, $\varphi \preceq \varphi'$ since $X \subset X'$. However, if conventional support is applied, the support of φ is 1 by $t_2 \mapsto t_a$, while the support of φ' is 2, since $(t_2, t_7) \mapsto (t_a, s_a)$ and $(t_2, t_9) \mapsto (t_a, s_a)$, violating anti-monotonicity. Intuitively, this is because in collective rules, a tuple can join with multiple tuples, yielding a larger "support". □

To fix this, we revise the notion of support and establish its anti-monotonicity, a property that is critical to reducing the search space when discovering rules. We assume *w.l.o.g.* that predicates in this section involve two tuple variables, *i.e.,* $t.A \oplus s.B$ or $\mathcal{M}(t[\bar{A}], s[\bar{B}])$; all notations extend naturally to predicates with one tuple variable.

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

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

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

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

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

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

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

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

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

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

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

Theorem 1: *For any instance $\mathcal{D}$ of $\mathcal{R}$ and* REEs *φ and φ', if $\varphi \preceq \varphi'$, then* $\text{spset}(\varphi', \mathcal{D}) \subseteq \text{spset}(\varphi, \mathcal{D})$ *and* $\text{supp}(\varphi', \mathcal{D}) \leq \text{supp}(\varphi, \mathcal{D})$. □

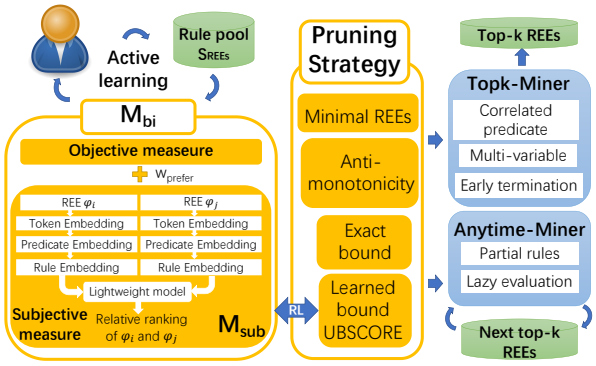


Figure 1: Workflow

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

Example 4: In Example 3, $\varphi \preceq \varphi'$. By Theorem 1, $\text{supp}(\varphi, \mathcal{D}) \geq \text{supp}(\varphi', \mathcal{D})$, since $\text{spset}(\varphi, \mathcal{D}) = \text{spset}(\varphi', \mathcal{D}) = \{t_2 \mapsto t_a\}$. □

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

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

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

We consider *minimal* REE φ on $\mathcal{D}$ that is (a) *non-trivial*: $p_0 \notin X$, and (b) *left-reduced*: φ is σ-frequent and δ-confident; moreover, there exists no σ-frequent and δ-confident REE φ' such that $\varphi' \preceq \varphi$.

Besides support and confidence, we also use *attribute diversity*, *minimality* and *succinctness* as objective measures, shown in [3] for the lack of space. Other objective measures can also be plugged in.

3.1.2 Subjective Measures. Different users have diverse preference for rules. Below we train an ML model $\mathcal{M}_{\text{sub}}$ for catching subjective user preference, to compensate objective ones. Similar to the objective measures that have bounds, *e.g.,* support and confidence, we design our model with bounded output values; as will be seen shortly, the bounds facilitate early termination in rule discovery.

Model architecture. As shown in Figure 1, we learn $\mathcal{M}_{\text{sub}}$ by first transforming each rule into an embedding, and then feeding it into a lightweight model, which outputs a scalar score, indicating the subjective preference of users. The details are explained as follows.

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

(1) Firstly, we embed each predicate p in X by tokenizing its operator and operands into three tokens, say T_1, T_2, and T_3, *e.g.,* if p is $t.A \oplus c$, we tokenize it as $t.A$, $\oplus$ and c; similar for other types of predicates. We treat tokens as words and construct a vocabulary $\mathcal{V}$. For each $T_i \in \mathcal{V}$, we use a pre-trained model, e.g., ELMo [75] or Bert [22], to transform T_i to a vector $\mathbf{T}_i \in \mathbb{R}^{d\times 1}$. Then we adopt a linear layer $\mathbf{w}_{\text{emb}} \in \mathbb{R}^{3\times 1}$ to generate the embedding $E_p \in \mathbb{R}^{d\times 1}$ of p:

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

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

(2) With the predicate embeddings for each p in X, we compute the *precondition embedding*, denoted by $E_X \in \mathbb{R}^{d_r\times 1}$ for X. Recall that X is a conjunction $p_1 \wedge \cdots \wedge p_{|X|}$ of predicates. The embedding E_X should be *permutation invariant*, *i.e.,* $E_X = E_{X'}$, where $X' = p_{\pi_1} \wedge \cdots \wedge p_{\pi_{|X|}}$, and $\pi_1, \ldots, \pi_{|X|}$ is a new permutation of $1, \ldots, |X|$. To achieve this, we adopt *deep sets* [98] based on the predicate embeddings, and obtain $E_X = \rho(\sum_{i=1}^{|X|} \Phi(E_{p_i}))$, where ρ and Φ are two linear layers without activation functions. Moreover, we use predicate embedding of p_0 as the consequence embedding, denoted by E_{p_0}.

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

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

Lightweight model. Given the rule embedding E_φ, our lightweight model employs a fully-connected layer along with a learnable parameter UB_{sub} and the ReLU activation function:

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

where $\mathbf{w}_{\text{light}} \in \mathbb{R}^{(d_r+d)\times 1}$ and $b_{\text{light}} \in \mathbb{R}$ are parameters of the lightweight model. Since we adopt the ReLU activation function, the subjective score is guaranteed to be no larger than UB_{sub}.

Example 5: Consider φ_3 in Example 1. We first tokenize its predicates, *e.g.,* $\{\mathcal{M}_{\text{new}}, t_a, s_b\}$ of $p = \mathcal{M}_{\text{new}}(t_a, s_b)$, and transform them into embeddings using, *e.g.,* ELMo. The representation of p is generated using $\mathbf{w}_{\text{emb}}$. After all predicates are embedded, we compute the precondition and consequence embeddings, and concatenate them to form E_{φ_3}. The subjective score is computed via $\mathcal{M}_{\text{sub}}$. □

Remark. Note that we do not adopt existing language models to acquire the rule-level representations for training $\mathcal{M}_{\text{sub}}$, for the following reasons: (1) REEs do not follow natural language structure and thus, directly applying language model (*e.g.,* Bert [22]) does not work well; and (2) given $\varphi : X \rightarrow p_0$, X is a conjunction of predicates, but not a sequence of predicates as in text for which there is no guarantee for the permutation invariant property.

3.2 Modeling User Preference

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

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

where w_f and w_g are non-negative weights associated with the measures. We assume *w.l.o.g.* that a larger value is more preferable for each measure. Denote the complete weight vector by $\mathbf{w}_{\text{prefer}}$, representing the user preference. Intuitively, a larger weight indicates that the corresponding measure is more important to the user. Once $\mathbf{w}_{\text{prefer}}$ is determined, we can compute the ranking score of each rule, based on which we can deduce the top-k rules.

As will be seen shortly, the weight vector and subjective model are *learned*. Users may also manually adjust $\mathbf{w}_{\text{prefer}}$. Domain experts may want to find unexpected rules to give novel insights, and can assign a larger weight to $\mathcal{M}_{\text{sub}}$. Novice users can turn off the subjective features by setting $w_g = 0$, and only use objective measures to find common rules as mined by conventional discovery algorithms. If users do not know exactly how to rank the rules, they could label a few rules, and our model (esp. G) can learn their preference.

As a real-life example, our drug-discovery users have accumulated quite a few rules for target identification [100], and the conventional notions of support and confidence do not help them find new rules. What they need is the subjective measure (*i.e., G*); hence they set w_f very small and let w_g dominate in $\mathbf{w}_{\text{prefer}}$. The subjective measure $\mathcal{M}_{\text{sub}}$ here suffices since it is learned via a neural network that has enough approximation power for users' preference.

3.3 An Active Learning Approach

Denote our bi-criteria model by $\mathcal{M}_{\text{bi}}$, which is the combination of subjective measures $\mathcal{M}_{\text{sub}}$, objective measures and $\mathbf{w}_{\text{prefer}}$. Below we show how to jointly learn $\mathcal{M}_{\text{bi}}$ via the *active learning* strategy.

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

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

Active learning. It is impractical for users to label all of rule pairs ($\frac{1}{2}|S_{\text{REEs}}| \times |S_{\text{REEs}}|$ in total). Thus we adopt active learning, which selects high-quality pairs of rules for users to label. We iteratively learn $\mathcal{M}_{\text{bi}}$ and actively select rule pairs in S_{REEs} that $\mathcal{M}_{\text{bi}}$ cannot distinguish well, *i.e.,* pairs that have the smallest differences in ranking scores, and ask users for labeling. The process proceeds until either it reaches the maximum number of iterations or the accuracy of $\mathcal{M}_{\text{bi}}$ reaches a predefined bound in a validation dataset. In the training step, we could simply combine different predicates and generate as many rules as required for S_{REEs} without worrying about the rule validity. We defer the details to [3] for the lack of space. We find that it often suffices for users to label 160 rule pairs in 5 rounds of interaction (80 in the first round). After training, $\mathcal{M}_{\text{bi}}$ is used to rank *valid* rules in the discovery process, as will be seen in Section 4.

4 TOP-K RULE DISCOVERY

Based on the bi-criteria model, we discover top-ranked rules from a dataset $\mathcal{D}$ (Section 4.1), and develop such an algorithm (Section 4.2).

4.1 Problem Statement

Denote by Σ_{all} the set of minimal REEs on $\mathcal{D}$ that are σ-frequent and δ-confident for thresholds σ and δ. As remarked earlier, Σ_{all} may

contain an excessive number of rules that are not very relevant to users' needs. To reduce such rules, we learn a *subset* $\mathcal{P}_0$ of predicates for each p_0, including all predicates correlated to p_0; we focus on mining REEs $\varphi: X \rightarrow p_0$ such that $X \subseteq \mathcal{P}_0$. We identify correlated attributes via graphical lasso [37] (see below) or LSTM [49].

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

- *Input*: A schema $\mathcal{R}$, an instance $\mathcal{D}$ of $\mathcal{R}$, the set $\mathcal{P}_{\text{all}}$ of all consequence predicates for discovery, the support/confidence thresholds σ/δ, an integer k, and the learned bi-criteria model $\mathcal{M}_{\text{bi}}$.
- *Output*: A set Σ of top-k REEs such that for each $\varphi: X \rightarrow p_0$ in Σ, (a) $p_0 \in \mathcal{P}_{\text{all}}$; (b) $X \subseteq \mathcal{P}_0$, $\mathcal{P}_0$ is a set of predicates correlated to p_0; and (c) φ is minimal, σ-frequent and δ-confident.

We impose lower bounds σ and δ on support and confidence to ensure that the discovered rules are valid and reliable, just like [74].

Workflow. Figure 1 shows the workflow. First, we train the bi-criteria model $\mathcal{M}_{\text{bi}}$ via active learning by interacting with the users using the rules in the rule pool $\mathcal{S}_{\text{REEs}}$. Given the learned $\mathcal{M}_{\text{bi}}$, we discover top-k rules and next k rules using algorithms Topk-Miner and Anytime-Miner, respectively, by adopting pruning strategies, *e.g.,* anti-monotonicity and upper bounding, for early termination. In particular, we employ an ML model UBSCORE, which takes the learned $\mathcal{M}_{\text{bi}}$ as inputs and outputs an estimation of the upper bound of ranking scores, via *reinforcement learning*, to be seen shortly.

4.2 A Top-k Discovery Algorithm

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

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

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

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

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

[P3] *Prune low-ranked* REEs: We maintain T_k, the k-th highest ranking score in Σ. We compute a score upper bound UB for rules expanded from φ. If UB is less than T_k, no rules expanded from φ can make a top-k rule and thus, we stop the exact expansion.

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

(1) Exact bound. We categorize the ranking measures into three types: *monotonic, anti-monotonic and general measures*: (a) A ranking measure h (*i.e.,* $f \in F$ or $g \in G$) is *monotonic* if $h(\varphi) \leq h(\varphi')$ as long as $\varphi \preceq \varphi'$, *i.e.,* adding predicates to φ monotonically increases the score. Then the upper bound (denoted by $h_{\text{ub}}(\varphi')$) of $h(\varphi')$ is $h(\mathcal{P}_0 \rightarrow p_0)$. (b) Adding predicates in an *anti-monotonic* measure monotonically decreases the ranking score, *e.g.,* support. Here the h_{ub} of $h(\varphi')$ is $h(\varphi)$. (c) If a ranking measure does not have the above properties, it is referred to as *general*, *e.g.,* usually the subjective model, whose h_{ub} is UB_{sub}. The exact upper bound of score(φ) is the weighted sum of h_{ub} of all measures (see [3] for details).

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

(2) Learned bound of subjective measures. Recall that $h_{\text{ub}} = \text{UB}_{\text{sub}}$ for subjective measures, which can be loose. Thus, we develop an ML model to learn a tighter bound, before the discovery process.

Given the model $\mathcal{M}_{\text{sub}}$ and a candidate REE $\varphi: X \rightarrow p_0$, we learn a function UBSCORE such that $\text{UBSCORE}(\varphi) \approx \max\{\mathcal{M}_{\text{sub}}(\varphi') \mid \varphi': X \cup P' \rightarrow p_0, \forall P' \subseteq \mathcal{P}_0\}$. One may want to learn UBSCORE via a regression model implemented as a feed-forward network (FFN), but it is exponential to label all training instances.

Therefore, we utilize Deep Q-learning (DQN) [69] to generate training instances of UBSCORE. It takes the currently selected predicates $\mathcal{P}_{\text{sel}}$ as state s, and the predicate p to be added as action a. In each step, the agent interacts with the environment, *i.e.,* $\mathcal{M}_{\text{sub}}$, to compute a reward $r = \mathcal{M}_{\text{sub}}(\mathcal{P}_{\text{sel}} \cup \{p\} \rightarrow p_0) - \mathcal{M}_{\text{sub}}(\mathcal{P}_{\text{sel}} \rightarrow p_0)$. We add a special action END to terminate the expansion of $\mathcal{P}_{\text{sel}}$, and also stop it if its length reaches a predefined constant. DQN contains two networks: a Q-network and a target network. The Q-network takes a state s and an action a as input, and outputs the reward of taking a. It is learned and updated in each step from s to s', where $s' = \mathcal{P}_{\text{sel}} \cup \{p\}$. The Q value is computed as

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

where γ is a discount ratio. The Q value is approximated by the Q-network, and the target network is obtained by cloning Q-network in a few steps. We denote the state s (*i.e.,* $\mathcal{P}_{\text{sel}}$) as a $|\mathcal{P}_0|$-dimensional bit vector $\mathbf{v}_s$, where $\mathbf{v}_s[p] = 1$ if $p \in \mathcal{P}_{\text{sel}}$, and $\mathbf{v}_s[p] = 0$ otherwise. We implement the Q-network as a feed-forward network and output a $(|\mathcal{P}_0| + 1)$-dimensional vector $\mathbf{v}_o$ of $|\mathcal{P}_0| + 1$ rewards, such that $\mathbf{v}_o[p]$ is the reward of adding p or terminating the expansion if p is END. The learning method and loss function are the same as DQN. We adopt the Double strategies [47] to speed up training.

After learning DQN, we generate N REEs as training instances for UBSCORE. We use the policy from DQN to obtain the subjective upper bounds as labels, by iteratively expanding $\mathcal{P}_{\text{sel}}$ with predicates of maximum rewards using DQN, until the termination condition is satisfied. When training instances are ready, we train UBSCORE to predict the upper bound of subjective measures.

Note that the training (including training DQN, label generation and training UBSCORE) does not dominate the complexity since (1) computing rewards in DQN is fast in $O(|\mathcal{M}_{\text{sub}}|)$ time; (2) DQN and UBSCORE are implemented as FFNs with a few hidden layers; and (3) both of the training episode in DQN and N are constants.

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

As shown in Figure 2, Topk-Miner is a *levelwise search* algorithm.

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

Topk-Miner adopts the following optimization strategies in rule discovery. (a) When multiple p_0 in $\mathcal{P}_{\text{all}}$ share similar correlated predicates $\mathcal{P}_0$, it processes these p_0 together (not shown). (b) It pre-computes auxiliary structures (line 2) such as position list indexes (PLI) [74] to efficiently compute supports and confidences.

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

Correlated predicate learning. Recall that for each consequence p_0, we learn a subset $\mathcal{P}_0$ of its correlated logic predicates and ML predicates. To this end, we maintain a pool of pre-trained ML models. Given a schema $\mathcal{R}$, we associate attributes in $\mathcal{R}$ to compatible models in the pool and initialize the ML predicates. Then, to learn correlated $\mathcal{P}_0$, we apply graphical lasso [37, 101] to learn how an attribute is affected by others. Informally, given a predicate p, either a logic or an ML predicate, if attributes in p have strong impact on the attributes in p_0, p is correlated to p_0 and is included in $\mathcal{P}_0$.

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

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

Algorithm Topk-Miner

Input: $\mathcal{D}$, $\mathcal{P}_{\text{all}}$, k, σ and δ.
Output: A heap Σ of top-k REEs such that for each $\varphi : X \rightarrow p_0$ in Σ, (1) $p_0 \in \mathcal{P}_{\text{all}}$; (2) $X \subseteq \mathcal{P}_0$, where $\mathcal{P}_0$ is a set of predicates correlated to p_0.

1. Σ := an empty max-heap of maximum size k, ordered by ranking scores;
2. Build auxiliary structures, *e.g.,* position list indexes (PLI) [74];
3. **for each** $p_0 \in \mathcal{P}_{\text{all}}$ **do**
4. $\quad \mathcal{P}_{\text{sel}} := \emptyset$; $\mathcal{P}_{\text{re}} := \mathcal{P}_0$;
5. $\quad \Sigma := \text{Expand}(\mathcal{D}, \mathcal{P}_{\text{sel}}, \mathcal{P}_{\text{re}}, p_0, k, \delta, \sigma, \Sigma)$;
6. **return** Σ;

Procedure Expand

Input: $\mathcal{D}$, $\mathcal{P}_{\text{sel}}$, $\mathcal{P}_{\text{re}}$, p_0, k, δ, σ and the current heap Σ of REEs.
Output: An updated heap Σ of REEs.

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

Figure 2: Algorithm Topk-Miner

We further optimize Topk-Miner by considering two effective processing orders for the predicates in $\mathcal{P}_{\text{re}}$ (line 18).

(1) Support-based processing order. The first order is to add the predicates p from $\mathcal{P}_{\text{re}}$ to $\mathcal{P}_{\text{sel}}$ based on $\text{supp}(\mathcal{P}_{\text{sel}} \wedge p, \mathcal{D})$, such that predicates with high supports are processed first. This helps us prune those predicates in $\mathcal{P}_{\text{re}}$ that are useless in generating σ-frequent REEs (in addition to **[P2]**). Intuitively, if including a predicate p with high support cannot lead to an σ-frequent REE, it is even more difficult for a predicate p' with low support to do so. In light of this, if $\mathcal{P}_{\text{re}}$ is ordered by support, every time we see a predicate p and if $\mathcal{P}_{\text{sel}} \wedge p \rightarrow p_0$ is not σ-frequent, we can prune all p' in $\mathcal{P}_{\text{re}}$ ordered after p with $\text{spset}(p', \mathcal{D}) \subseteq \text{spset}(p, \mathcal{D})$ (see [3] for a proof).

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

Example 7: Consider the relations in Table 1. Assume that $\mathcal{P}_{\text{sel}} = \{\text{Drug}(t)\}$ and p_0 is t.weight $\neq$ 0. Let p and p' be t.formula = "$C_{10}H_{15}N_5$" and t.name = "Phenformin", respectively. Clearly, $\text{spset}(p', \mathcal{D}) = \{t_2 \mapsto t\}$ is a subset of $\text{spset}(p, \mathcal{D}) = \{t_2 \mapsto t, t_4 \mapsto t\}$ and thus, p is processed before p'. If we find that $\mathcal{P}_{\text{sel}} \wedge p \rightarrow p_0$ is not σ-frequent, there is no need to process p', since $\text{supp}(\mathcal{P}_{\text{sel}} \wedge p \rightarrow p_0) = 2 > 1 = \text{supp}(\mathcal{P}_{\text{sel}} \wedge p' \rightarrow p_0)$. □

(2) Score-based order. We can also process the predicates in $\mathcal{P}_{\text{re}}$ based on their "potential" in mining rules with high ranking scores. Intuitively, if more rules with high scores are found, the score bound T_k (*i.e.,* the k-th highest one observed so far) is tighter and thus, more rules are likely to be pruned by **[P3]** at an earlier stage.

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

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

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

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

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

Example 8: We show how φ_2 in Example 1 is found. Assume that $\mathcal{P}_{\text{sel}} = \{\text{Drug}(t_a), \text{Disease}(s_a), \text{Disease}(s_b)\}$, p_0 is $t_a.\text{cid} = s_b.\text{cid}$, and $\mathcal{P}_{\text{re}} = \{t_a.\text{cid} = s_a.\text{cid}, \mathcal{M}_{\text{therapy}}(s_a, s_b)\}$. We expand $\mathcal{P}_{\text{sel}}$ by adding predicates in $\mathcal{P}_{\text{re}}$ one by one. Before we perform the exact expansion, we first compute the upper bound, say UB. If UB is less than T_k, we terminate early since expanding $\mathcal{P}_{\text{sel}}$ will not give any top-k REEs **[P3]**. Otherwise, assume that $t_a.\text{cid} = s_a.\text{cid}$ has a larger $\text{supp}(\mathcal{P}_{\text{sel}} \wedge p)$. Then we add it to $\mathcal{P}_{\text{sel}}$ first, by the support-based processing order, until we find that φ_2 is a minimal REE. If φ_2 has a higher score than some rules in Σ, Σ is updated accordingly. □

Complexity analysis. Topk-Miner takes $O(\sum_{\varphi \in C(\mathcal{P}_0) \times \mathcal{P}_{\text{all}}} |\mathcal{D}|^{|\varphi|})$ time in the worst case, where $C(\mathcal{P}_0)$ is the power set of $\mathcal{P}_0$ and $|\varphi|$ is the number of predicates in φ, since Topk-Miner examines the entire $C(\mathcal{P}_0)$ for each $p_0 \in \mathcal{P}_{\text{all}}$ at worst. We will parallelize Topk-Miner in Section 6 to scale with large datasets.

5 ANYTIME DISCOVERY

We convert Topk-Miner into an anytime algorithm Anytime-Miner, such that we can get next top-k rules if needed, via lazy evaluation.

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

Denote by Σ (see Figure 2) the heap of top-k REEs discovered so far. We expand Σ to *lazily* discover the next top-k results as follows.

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

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

(3) We ensure that the next top-k results are not "redundant", *i.e.,* not logical consequences of the rules that have been shown before. Thus, we apply the implication analysis [32] on each newly discovered rules. Formally, we say that a set of REEs Σ *entails* another REE φ over $\mathcal{R}$, denoted by $\Sigma \models \varphi$, if for any instance $\mathcal{D}$ of $\mathcal{R}$, if $\mathcal{D} \models \Sigma$ then $\mathcal{D} \models \varphi$. Then, every time a complete rule φ is discovered, we add it to the heap Σ only if $\Sigma \not\models \varphi$. While the implication problem is Π_2^p-complete [33], we develop an efficient heuristic for checking.

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

6 PARALLEL TOP-k RULE DISCOVERY

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

6.1 Parallel Scalability

We revisit the widely adopted notion of parallel scalability [55].

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

$$T(|\mathcal{D}|, |\mathcal{P}_{\text{all}}|, \sigma, \delta) = \widetilde{O}\left(\frac{t(|\mathcal{D}|, |\mathcal{P}_{\text{all}}|, \sigma, \delta)}{n}\right),$$

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

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

6.2 Parallel Algorithm

We next parallelize Topk-Miner and develop PTopk-Miner (Figure 3). PTopk-Miner runs with one coordinator S_c and n workers $P_1, \ldots, P_n$ under the Bulk Synchronous Parallel (BSP) model [92], where the coordinator is responsible for distributing and balancing workloads, and workers discover rules in parallel. The overall computation is divided into supersteps of a fixed duration.

Overview. Same as Topk-Miner, the coordinator maintains a max-heap of maximum size k, consisting of the top-ranked REEs discovered so far (line 1). Denote the heap at superstep i by Σ_i, and the k-th highest ranking score in Σ_i by T_k^i. The coordinator first distributes the workloads evenly to all workers (see below; line 2-5). Then, each worker parallelly processes its workload and discovers rules in supersteps (line 6-15). At each superstep, the coordinator informs each worker the latest score bound T_k^i (line 10), based on

Algorithm PTopk-Miner

Input: $\mathcal{D}$, $\mathcal{P}_{\text{all}}$, k, σ, γ, a coordinator S_c and n workers $P_1, \ldots, P_n$.
Output: A max-heap Σ of top-k REEs on $\mathcal{D}$.

/* *executed at coordinator* S_c */
1. $i := 0$; $\Sigma_i :=$ an empty max-heap of maximum size k;
2. **for each** $p_0 \in \mathcal{P}_{\text{all}}$ **do**
3. Construct a work unit $w = \langle \mathcal{P}_{\text{sel}}, \mathcal{P}_{\text{re}}, p_0 \rangle$, where $\mathcal{P}_{\text{sel}} = \emptyset$ and $\mathcal{P}_{\text{re}} = \mathcal{P}_0$;
4. Evenly divide $\mathcal{P}_{\text{all}}$ into n partitions, namely $\text{RHS}_1, \ldots, \text{RHS}_n$;
5. Assign workload $\mathcal{W}_j = \{w = \langle \mathcal{P}_{\text{sel}}, \mathcal{P}_{\text{re}}, p_0 \rangle \mid p_0 \in \text{RHS}_j\}$ to worker P_j;

/* *run on* n *workers in parallel, in supersteps* */
6. **for each** worker P_j **do**
7. Fetch $\mathcal{D}_{\mathcal{W}_j} = \{t \in \mathcal{D} \mid \exists s \in \mathcal{D}, p \in P_0 \text{ s.t. } h\langle t, s\rangle \models p \text{ or } h\langle t, s\rangle \models p_0,$ where $p_0 \in \text{RHS}_j\}$ and build the corresponding auxiliary structures;
8. **while** there exists unfinished work **do** /* *superstep* i */
9. **for each** P_j with non-empty workload W_j **do**
10. $\mathsf{T}_k^i :=$ the k-th highest ranking score in Σ_i (informed by S_c);
11. Run Topk-Miner at P_j based on $\mathcal{W}_j$ and $\mathcal{D}_{\mathcal{W}_j}$ in parallel;
12. S_c pulls top-k REEs φ newly discovered (score$(\varphi) > \mathsf{T}_k^i$);
13. **for each** P_x that has finished the assigned workload **do**
14. Balance workload between P_x and the heaviest worker P_j;
15. Upon receiving new REEs from workers, S_c updates Σ_i to Σ_{i+1}; update T_k^i to T_k^{i+1}; broadcast T_k^{i+1} to all workers; $i := i + 1$;
16. **return** Σ_i;

Figure 3: Algorithm PTopk-Miner

which each worker performs the subsequent discovery (line 11) by applying the pruning strategies in Section 4.2. The coordinator S_c pulls the newly discovered top-k rules from each worker at the end of each superstep (line 12). In addition, it adjusts and balances the workload when needed (line 13-14; see below). Moreover, S_c extends the heap Σ_i to Σ_{i+1} with the new rules, and updates score bound T_k^i to T_k^{i+1} (line 15). The process continues until all workers finish, *i.e.,* when no rules with scores above the bound can be found.

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

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

Upon receiving $\mathcal{W}_j$, worker P_j fetches a subset $\mathcal{D}_{\mathcal{W}_j}$ of data from $\mathcal{D}$, guided by $\mathcal{W}_j$, where $\mathcal{D}_{\mathcal{W}_j} = \{t \in \mathcal{D} \mid \exists s \in \mathcal{D}, p \in \mathcal{P}_0 \text{ s.t. } h\langle t, s\rangle \models p \text{ or } h\langle t, s\rangle \models p_0, \text{ where } p_0 \in \text{RHS}_j\}$; it also constructs the corresponding auxiliary structures for performing rule discovery. In this way, the same data will be merged and transmitted to P_j only once even if it satisfies multiple predicates, reducing the total communication cost when processing multiple predicates.

Example 10: Consider Drug in Table 1. Let p be t.type = s.type and p' be t.formula = s.formula. Assume that coordinator S_c assigns workload $\mathcal{W}_j = \{w, w'\}$ to worker P_j, where $w = \langle \emptyset, \{p\}, p_0 \rangle$ and $w' = \langle \emptyset, \{p'\}, p_0' \rangle$. It is easy to see $\mathcal{D}_{\mathcal{W}_j} = \{t_2, t_3, t_4\}$. In particular, although $h\langle t_4, t_3\rangle \models p$ and $h\langle t_4, t_2\rangle \models p'$, t_4 is transmitted once. □

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

Name	Type	#tuples	#attributes	#relations
Adult [54, 65, 74]	real-life	32,561	15	1
Airport [65, 74]	real-life	55,113	18	1
DrugDisease [21]	real-life	466,658	35	4
Inspection [65, 74, 79]	real-life	170,000	19	1
NCVoter [54, 65, 74]	real-life	1,681,617	12	1
DBLP [90]	real-life	1,799,559	18	3
Tax [14, 20, 28, 65, 74]	synthetic	10,000,000	15	1

Table 3: Dataset statistic

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

Parallel scalability. The parallel scalability is shown as follows.

Theorem 3: *Algorithm* PTopk-Miner *is parallelly scalable relative to the sequential algorithm* Topk-Miner. □

Proof. Recall the complexity of Topk-Miner is $t(|\mathcal{D}|, |\mathcal{P}_{\text{all}}|, \sigma, \delta) = O(\sum_{\varphi \in C(\mathcal{P}_0) \times \mathcal{P}_{\text{all}}} |\mathcal{D}|^{|\varphi|})$. We next show that the parallel runtime of PTopk-Miner is in $O(\frac{t(|\mathcal{D}|, |\mathcal{P}_{\text{all}}|, \sigma, \delta)}{n})$. In PTopk-Miner, S_c conducts workload assignment and maintains the global top-k by collecting REEs from each worker. The former takes $O(|\mathcal{P}_{\text{all}}|)$ time, while the latter takes $O(nk\log(k))$ time using merge-sort. Both are smaller than the discovery cost, which dominates the complexity.

The cost at each worker is dominated by the following: (a) transmit its top-k rules to the coordinator in time much less than $O(|\mathcal{D}|)$ since each rule is discovered from $\mathcal{D}$ and k is a small number; (b) receive T_k from S_c in $O(1)$ time; (c) balance its workload, where $O(|\mathcal{D}|)$ data is sent to idle workers; and (d) locally perform discovery in $O(\frac{t(|\mathcal{D}|, |\mathcal{P}_{\text{all}}|, \sigma, \delta)}{n})$ time, since the workload is evenly distributed by (c). Taken together, the parallel cost of PTopk-Miner is $O(\frac{t(|\mathcal{D}|, |\mathcal{P}_{\text{all}}|, \sigma, \delta)}{n})$ in the worst-case. In practice, the pruning strategies in Section 4.2 effectively remove useless candidates. □

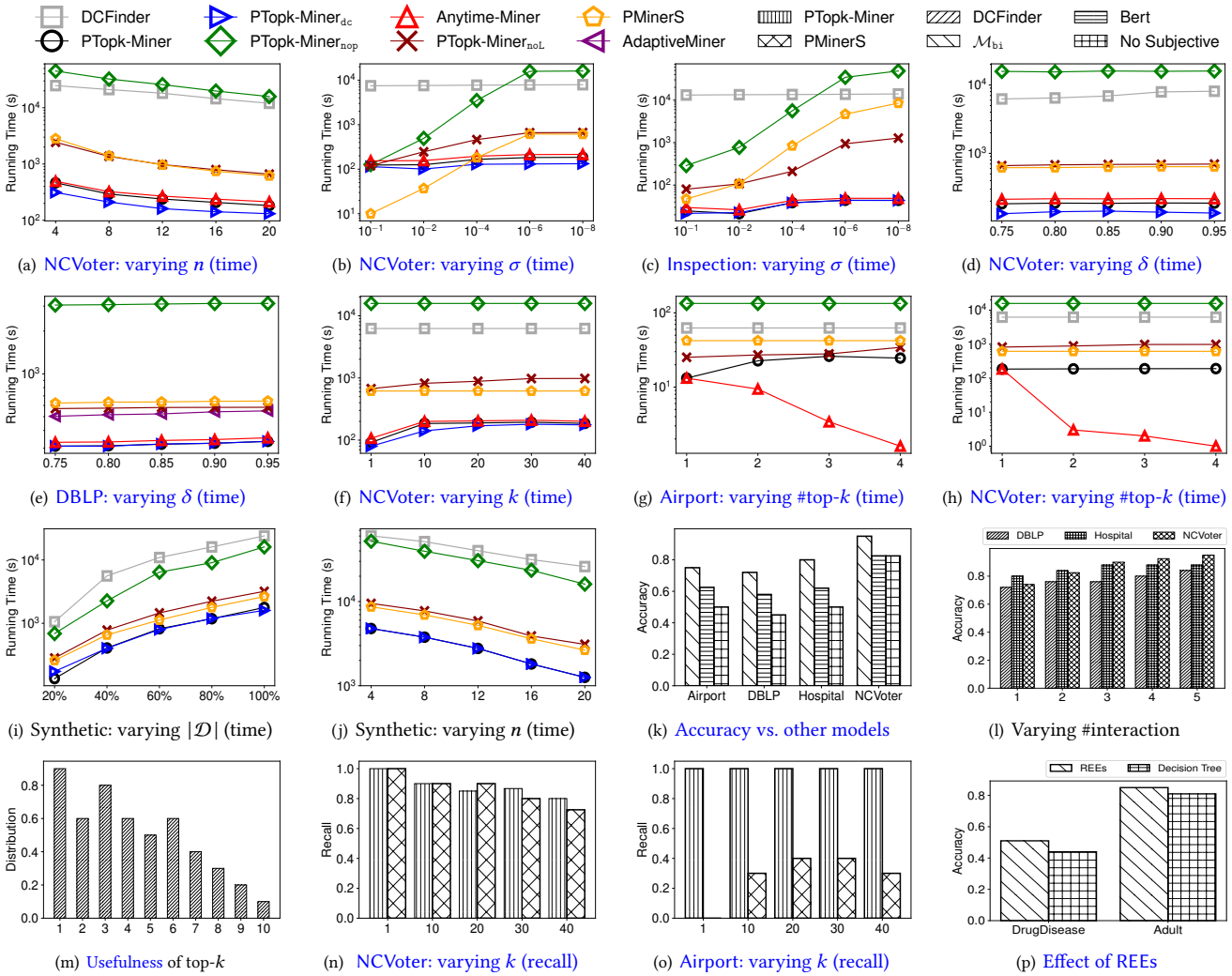
7 EXPERIMENTAL STUDY

Using real-life and synthetic data, we evaluated (1) the scalability of PTopk-Miner for top-k discovery and Anytime-Miner for anytime discovery, (2) the effectiveness of the bi-criteria model, (3) the accuracy of top-k discovery, (4) the effectiveness of top-k discovery.

Experimental setting. We start with the experimental setting.

Datasets. Following the setting in studies [65, 74], we used seven datasets, shown in Table 3. Adult, Airport, DrugDisease, Inspection, and NCVoter are real-life datasets commonly used in the literature. We additionally used an academic dataset DBLP that has multiple relations, and a synthetic dataset Tax that is obtained by first duplicating tuples of the original tax data (1M) [14, 20] 10 times and then modifying their attributes using a program of [28].

ML models. We used three ML predicates in REEs: (a) SentenceBert [78] for textual attributes where traditional logic predicates do not work well; (b) ditto [61] for ER, and (c) a model based on Bert [22] for assessing DDA (drug-disease association) in DrugDisease. For $\mathcal{M}_{\text{bi}}$, we used the original Bert [22] to initialize token embeddings and set the rule embedding to 100. Token embeddings are also



(a) NCVoter: varying n (time) (b) NCVoter: varying σ (time) (c) Inspection: varying σ (time) (d) NCVoter: varying δ (time)

(e) DBLP: varying δ (time) (f) NCVoter: varying k (time) (g) Airport: varying #top-k (time) (h) NCVoter: varying #top-k (time)

(i) Synthetic: varying $|\mathcal{D}|$ (time) (j) Synthetic: varying n (time) (k) Accuracy vs. other models (l) Varying #interaction

(m) Usefulness of top-k (n) NCVoter: varying k (recall) (o) Airport: varying k (recall) (p) Effect of REEs

Figure 4: Performance evaluation

learned together during training. For UBSCORE, we adopted FFN with three 200-dimensional hidden layers. We used Adam optimizer to train $\mathcal{M}_{bi}$ (resp. UBSCORE) with a batch-size of 128 (resp. 64), and the learning rate is 0.001 (resp. 0.0001). We adopted 300 epochs on Tesla V100 GPU. The inferences of $\mathcal{M}_{bi}$ and UBSCORE during rule discovery are re-implemented using the EJML library [2].

Baselines. We implemented the following, all in Java: (1) PTopk-Miner. (2) Anytime-Miner. (3) PTopk-Miner$_{nop}$, a variant of PTopk-Miner that mines all rules, sorts them by ranking scores and returns the top-k ones. (4) PTopk-Miner$_{noL}$, a variant of PTopk-Miner without the learned bound UBSCORE. (5) PTopk-Miner$_{dc}$, another variant to discover DCs, a special case of REEs on a single table without ML predicates. (6) DCFinder [74], a DCs discovery algorithm, which mines bi-variable DCs only; as shown in [74], it outperforms other DCs discovery methods; we parallelize it using [83] and extend it to support constant predicates and ML models by adding them into the evidence set used by [74]. (7) AdaptiveMiner [77], a MapReduce algorithm for mining association rules. (8) PMinerS [31], a parallel REE discovery method by sampling. We compared with PTopk-Miner$_{nop}$ and PTopk-Miner$_{noL}$ to test the effectiveness of pruning strategies and the learned bound, and with the others for efficiency although DCs and association rules are restricted REEs.

We conducted experiments on a cluster of up to 21 virtual machines (one for the coordinator), each powered by 64GB RAM and 18 processors with 3.10 GHz. We ran the experiments 3 times, and report the average here. We do not include the time of loading data and constructing auxiliary structure, *e.g.*, PLI for all algorithms. The bi-criteria model $\mathcal{M}_{bi}$ was trained via active learning once offline.

Experimental results. We next report our findings.

Exp-1: Scalability test. We evaluated the scalability of PTopk-Miner and Anytime-Miner. Unless stated explicitly, the default setting is $n = 20$, $\sigma = 10^{-6} \cdot |\mathcal{D}|^2$, $\delta = 0.75$, $k = 10$, and the default number of tuple variables in REEs is set to 2, for a fair comparison with DCs discovery that are restricted to the bi-variable setting. We adopted the combined predicate processing order and used 3 objective measures and 1 subjective measure. For the lack of space, we mainly show the results on NCVoter, one of the largest real-life dataset; the results on the other datasets are consistent.

Varying n. We varied the number n of machines from 4 to 20. As shown in Fig. 4(a), (a) PTopk-Miner scales well with the increase of machines: it is 3.15 times faster when n varies from 4 to 20. (b) It is feasible in practice. It takes 183s on NCVoter when $n = 20$, as opposed to 12,003s by DCFinder. (c) PTopk-Miner is 99.40 and 4.29

times faster than PTopk-Miner$_{\text{nop}}$ and PTopk-Miner$_{\text{noL}}$ on average, up to 110.65 and 5.25 times, respectively. This verifies the effectiveness of our pruning strategies and the learned bound UBSCORE. (d) PTopk-Miner is 67.24 times faster than DCFinder on average, up to 75.80 times, which demonstrates the advantage of top-k discovery, even though it discovers more expressive REEs. (e) Although PMinerS is executed in the small samples, it is 4.37 slower than PTopk-Miner on average, which further verifies the effectiveness of the pruning strategies. We will discuss the accuracy in Exp-3. (e) Although PTopk-Miner$_{\text{dc}}$ is slightly faster than PTopk-Miner, it only discovers DCs, a special case of REEs. (f) Anytime-Miner is slightly slower than PTopk-Miner, since it has to maintain more rules in the heap (Section 5). Nonetheless, its advantage is evident when users continuously want next top-k results, as will be seen shortly.

Varying σ. Varying the support threshold σ from $10^{-1}|\mathcal{D}|^2$ to $10^{-8}|\mathcal{D}|^2$, we report the results in Figures 4(b) and 4(c). As expected, all algorithms take longer when σ is smaller since they examine more candidates, *e.g.,* PTopk-Miner$_{\text{nop}}$ is 147.57 times slower when σ changes from $10^{-1}|\mathcal{D}|^2$ to $10^{-8}|\mathcal{D}|^2$. Nevertheless, PTopk-Miner is faster than PTopk-Miner$_{\text{nop}}$, PTopk-Miner$_{\text{noL}}$ and DCFinder under all σ, consistent with Fig. 4(a). DCFinder is not sensitive to σ because it spends most of the time on evidence set construction, which is not related with σ. PTopk-Miner is also less sensitive to σ, since it checks less REEs than PTopk-Miner$_{\text{nop}}$ due to its pruning strategies. Anytime-Miner has a similar trend as PTopk-Miner.

Varying δ. We varied the confidence bound δ from 0.75 to 0.95. As shown in Figure 4(d), (a) most algorithms are faster given a smaller δ, *e.g.,* PTopk-Miner and Anytime-Miner are 1.02 times and 1.11 times faster, when δ varies from 0.95 to 0.75. This is because higher δ indicates REEs with fewer violations, and hence more REEs are checked. (b) PTopk-Miner consistently outperforms the baselines. (c) PTopk-Miner also performs the best for association rules (essentially constant CFDs). Figure ?? shows the discovery time to mine such rules on DBLP, and we set $\sigma = 10^{-4}|\mathcal{D}|$. PTopk-Miner is 1.64 times faster than AdaptiveMiner on average.

Varying k. Varying k from 1 to 40, Figure 4(e) shows that PTopk-Miner$_{\text{nop}}$, PMinerS and DCFinder are indifferent to the value of k, since they mine all rules (*i.e.,* REEs or DCs) from the dataset regardless of k. In contrast, PTopk-Miner and PTopk-Miner$_{\text{noL}}$ take much less time than the four due to its pruning strategies for top-k discovery; their costs increase when k gets larger since more REEs have to be checked. Anytime-Miner is also faster than the four, by adopting lazy evaluation for skipping unnecessary REEs expansions. PMinerS is 4.05 times slower than PTopk-Miner on average, which shows the effectiveness of its pruning strategies.

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

Results are reported in Figures 4(f) and 4(g), when varying #top-k from 1 and 4. For the first top-k REEs, there is no big difference in the runtime between Anytime-Miner and PTopk-Miner. However, the advantage of Anytime-Miner over PTopk-Miner is more evident when the users want to see more top-k results, *e.g.,* when one asks for the 3rd top-10 REEs, Anytime-Miner is 95.0 times and 10.57 times faster then PTopk-Miner on NCVoter and Inspection, respectively. This is because Anytime-Miner maintains partial results and is more efficient to resume the discovery. If Anytime-Miner accumulates sufficient partial results (including rules that have been discovered but not in the top-k list), *e.g.,* the 1st to the 3rd top-10 on NCVoter, its runtime may decrease rapidly because most of rules in next #top-k are pre-computed and stored. However, when partial results are not enough, Anytime-Miner needs to spend time to discover more satisfied ones, *e.g.,* #top-k from 1 to 2 on Airport.

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

Varying $|\mathcal{D}|$ (synthetic). We varied the scaling factor of $\mathcal{D}$ from 20% to 100%, *i.e.,* we changed the number of relations and tuples per relation from 2 million to 10 million. As shown in Figure 4(h), all algorithms take longer, as expected. PTopk-Miner still outperforms all competitors including PMinerS. It takes 0.5h when $\mathcal{D}$ has 10M tuples, as opposed to 0.9h, 4.4h, 0.74h and 6.5h by PTopk-Miner$_{\text{noL}}$, PTopk-Miner$_{\text{nop}}$, PMinerS and DCFinder, respectively.

Varying n (synthetic). Fixing $|\mathcal{D}|$ as 10M, we varied the number n of machines from 4 to 20 in Figure 4(i). Consistent with Figures 4(a). PTopk-Miner is 3.60 times faster when n varies from 4 to 20.

Exp-2: Effectiveness of the bi-criteria model. In this set of experiments, we studied the effectiveness of our bi-criteria model. Domain experts (our industry partner) labeled 400 pairs of rules, who well understood requirements of multiple users. The rule pairs are split into training, validation and testing sets as 80%, 10% and 10%. The accuracy in the testing data is measured by the percentage of rule pairs whose relative rank is correctly identified.

Accuracy vs. language models. We compared our bi-criteria model against Bert [22], a state-of-the-art language model that is implemented as a binary classifier with pairs of rules as inputs, separated by [SEP], such that a softmax layer is added after the embedding of [CLS]. The model is fine-tuned. Here we use the bert_en_uncased version. As reported in Fig. 4(j), our model consistently beats Bert in accuracy on all datasets, *e.g.,* on NCVoter, our accuracy is 0.95, 13% higher than Bert. This is due to unique features such as permutation invariant of logic rules, which cannot be captured by existing language models that learn rules as natural language.

Effectiveness of subjective measures. We next studied the effectiveness of subjective measures by comparing the accuracy of our model with and without subjective measures. Here subjective measures are learned in Section 3.1. The results in Fig 4(j) verify that objective measures alone cannot meet the users' needs well since different users have diverse preference, *e.g.,* on DBLP, introducing subjective measures improves the accuracy by 30%, which shows the effectiveness of subjective measures in rule discovery.

Varying #interaction. To verify the usefulness of active learning, we report the accuracy by varying the number #r of rounds of interaction in Figure 4(l). Our industry partners labeled 20 pairs of rules in each round after 80 initial ones, a workload acceptable to domain experts. With larger #r, the accuracy increases, *e.g.,* after 5 rounds, the accuracy changes from 0.75 to 0.95 on NCVoter.

Moreover, after 5 rounds, the accuracy gets stable since the model has accumulated enough training data. We find that it typically requires 5 rounds of interactions to achieve a stable accuracy, and the improvement is substantial (*e.g.,* 10% on DBLP) with only 80 extra labeled instances by the active learning approach.

Usefulness of top-k REEs. Different experts have diverse preference, even when they are given the same pair of rules. To evaluate whether our top-k discovery ranks rules reasonably, we set up 10 *independent* tasks over five datasets (Airport, DBLP, Inspection, NCVoter, and Adult); on each of them, we mined top-10 REEs using two different consequence sets. Note that each task has its unique purpose, *e.g.,* Adult with consequence t.class = "$\leq$ 50K" aims to find rules for identifying the factors of low salary. For each task, we asked one expert to label 5 rules among the top-10 that s/he thinks are useful. This is to mitigate the effect of conflicting preference of different experts. In Figure 4(m), we plot the percentage of their "votes" for these 10 tasks. It shows that rules ranked higher by PTopk-Miner are more favored. More specifically, the rules ranked first to fifth by PTopk-Miner are labeled as useful by 68% of users on average, as opposed to 32% for the rules ranked sixth to tenth. The user ranking justifies the semantic of top-k discovery.

Exp-3: Accuracy test. We evaluated the accuracy of top-k discovery. Note that PTopk-Miner$_{\text{nop}}$ returns exact top-k. PMinerS mines all rules on samples, from which we get the top-k for comparison.

Recall. Varying k from 1 to 40, we reported the recall in Figures 4(n) and 4(o). Recall is defined as $\frac{|\Sigma_M \cap \Sigma_{GT}|}{k}$, where Σ_M (resp. Σ_{GT}) is the set of top-k rules discovered by the method M, *e.g.,* PTopk-Miner and PMinerS (resp. PTopk-Miner$_{\text{nop}}$). The recall of PTopk-Miner is close to PTopk-Miner$_{\text{nop}}$, and it is 1.0 for all k in Airport. This verifies that using UBSCORE for pruning incurs little loss. PTopk-Miner$_{\text{noL}}$ returns the exact top-k as PTopk-Miner$_{\text{nop}}$, since the exact bound does not miss any rules. PMinerS is not accurate, *i.e.,* 0.3, as opposed to 1.0 by PTopk-Miner on Airport. We do not evaluate precision because it is the same as recall in this setting.

Exp-4: Effectiveness of PTopk-Miner**.** Finally, we evaluated our effectiveness on labeled datasets and present the case study.

Effectiveness. We evaluated the accuracy of PTopk-Miner and decision tree model on the labeled datasets DrugDisease and Adult, for predicting the half-life time of drugs (multi-classification) and adult incomes (binary classification) in Fig. 4(p). For drugs, we transform scalar values of half_life_hours_curated attribute to 5 categories. We split data into training, validation and testing sets with 80%, 10% and 10%. PTopk-Miner is effective; it is 5.5% more accurate than decision tree on average, respectively, up to 7%.

Case study. We manually checked REEs discovered by PTopk-Miner on DrugDisease and Inspection, with $\sigma \geq 10$ and $\delta \geq 0.85$.

(1) DrugBank(t_0) $\wedge$ DiseaseMeSH(t_1) $\wedge$ Map(t_2) $\wedge$ Map(t_3) $\wedge$ t_1.PreferredConceptYN = Y $\wedge$ t_1.ConceptPreferredTermYN = Y $\wedge$ $\mathcal{M}_{\text{bert}}(t_0.\bar{A}, t_1.\bar{B})$ $\wedge$ t_0.id = t_2.drug_id $\wedge$ t_1.id = t_3.disea_id $\rightarrow$ t_2.id = t_3.id, where $\bar{A}$ and $\bar{B}$ are all attributes of t_0 and t_1. The rule states that if one disease with preferred concept and term is semantically close to a drug (predicted by Bert), they have drug-disease association (DDA), as indicated by the same mapping tuple in Map. This rule ranks high since our drug-discovery collaborators want to find new DDA rules that involve both drugs and diseases.

(2) DrugBank(t_0)$\wedge$DrugBank_halflife(t_1)$\wedge t_0$.cal_water_solubility = C1 $\wedge$ t_0.cal_logp = C2 $\wedge$ t_0.cal_molecular_weight = C3 $\wedge$ t_0.id = t_1.id $\rightarrow$ t_1.half_life_hours_curated = C4, where C1-4 are intervals C1 = $(-4, 1)$, C2 = $(0, 1)$, C3 = $(250, 350)$ and C4 = $(0, 15)$, and cal_water_solubility, cal_logp, cal_molecular_weight are three features of drugs, for the aqueous solubility, the predicted partition coefficient, and the predicted ratio of the average mass [1], respectively. This rule says that if the three features fit the ranges, then it takes less than 15 hours for the amount of this drug in the body to be reduced by one half. Such rules are prioritized by a pharmaceutical user who wants to study the factor of the short half-life time.

(3) Inspection(t_0) $\wedge$ Inspection(t_1) $\wedge$ t_0.AKA_Name = t_1.AKA_Name $\wedge$ t_0.inspection_Type = Canvass $\wedge$ t_1.Results = Pass $\rightarrow$ t_0.Risk = t_1.Risk. This rule says that two firms are equally risky if they have the same names and the inspection type of one firm is Canvass and the inspection result of the other firm is Pass. The rule ranks high for inspectors who assess the risk level of regulated facilities to determine a firm's compliance with laws and regulations.

Summary. We find the following. (1) Top-k discovery speeds up PTopk-Miner$_{\text{nop}}$, PTopk-Miner$_{\text{noL}}$ and DCFinder by 134, 6 and 86 times on average, respectively, up to 168, 12 and 138 times. When $n = 20$, it takes less than 200s to mine top-10 REEs from NCVoter that has 1.68M tuples, as opposed to 15,798s, 661s and 12,003 by PTopk-Miner$_{\text{nop}}$, PTopk-Miner$_{\text{noL}}$ and DCFinder. (2) PTopk-Miner scales well with parameters σ, δ and k. (3) PTopk-Miner is parallelly scalable: on average, it is 3.12 times faster when the number n of machines varies from 4 to 20. (4) The lazy evaluation strategy of Anytime-Miner is effective: Anytime-Miner is 52.8 times faster than PTopk-Miner when the users want the 4th top-10 REEs. (5) Our pruning strategies and the learned bound UBSCORE are effective, *e.g.,* reducing the the runtime of PTopk-Miner by 12.79 and 5.19 times on the Tax data of 10M tuples. (6) Our bi-criteria model is on average 15% more accurate than language models. In particular, the subjective measures in the model improves the accuracy from 0.525 to 0.76. (7) PTopk-Miner is capable of finding useful REEs from real-life data, beyond DCs by DCFinder and association rules by AdaptiveMiner, which are just special cases of REEs.

8 CONCLUSION

We have studied discovery of top-k rules. Our novelty consists of the following: (1) a bi-criteria model with both objective and subjective measures; (2) an active-learning method to learn the subjective model and weight vector of various measures; (3) a top-k algorithm for discovering REEs, which subsume common data quality rules and association rules as special cases; (4) an anytime algorithm to continuously mine the next top-k rules via lazy evaluation, and (5) parallelization of the algorithms with the parallel scalability. Our experimental study has shown that the methods are promising.

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

REFERENCES

[1] [n.d.]. Drugbank. https://docs.drugbank.com/xml/#external-codes.
[2] [n.d.]. EJML library. http://ejml.org/wiki/index.php?title=Main_Page.
[3] 2021. Full version. *https://github.com/yyssl88/topkdiscovery/blob/main/paper.pdf*.
[4] Ziawasch Abedjan, Patrick Schulze, and Felix Naumann. 2014. DFD: Efficient functional dependency discovery. In *CIKM*. 949–958.
[5] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases*. Addison-Wesley.
[6] Christopher P Adams and Van V Brantner. 2006. Estimating the cost of new drug development: is it really 802 million? *Health affairs* 25, 2 (2006), 420–428.
[7] Rakesh Agrawal and Ramakrishnan Srikant. 1994. Fast Algorithms for Mining Association Rules in Large Databases. In *VLDB*, Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo (Eds.).
[8] Chidanand Apté and Sholom M. Weiss. 1997. Data mining with decision trees and decision rules. *Future Gener. Comput. Syst.* 13, 2-3 (1997), 197–210.
[9] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. 1999. Consistent Query Answers in Inconsistent Databases. In *PODS*.
[10] Zeinab Bahmani and Leopoldo E. Bertossi. 2017. Enforcing Relational Matching Dependencies with Datalog for Entity Resolution. In *FLAIRS*.
[11] Nurken Berdigaliyev and Mohamad Aljofan. 2020. An overview of drug discovery and development. *Future Medicinal Chemistry* 12, 10 (2020), 939–947.
[12] Leopoldo E. Bertossi, Solmaz Kolahi, and Laks V. S. Lakshmanan. 2013. Data Cleaning and Query Answering with Matching Dependencies and Matching Functions. *Theory Comput. Syst.* (2013).
[13] Indrajit Bhattacharya and Lise Getoor. 2007. Collective entity resolution in relational data. *TKDD* (2007).
[14] Tobias Bleifuß, Sebastian Kruse, and Felix Naumann. 2017. Efficient Denial Constraint Discovery with Hydra. *PVLDB* 11, 3 (2017), 311–323.
[15] Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur. 1997. Dynamic Itemset Counting and Implication Rules for Market Basket Data. In *SIGMOD*, Joan Peckham (Ed.). ACM Press, 255–264.
[16] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).
[17] Paul Suganthan G. C., Adel Ardalan, AnHai Doan, and Aditya Akella. 2018. Smurf: Self-Service String Matching Using Random Forests. *PVLDB* 12, 3 (2018), 278–291.
[18] Chun-Chieh Chen, Chi-Yao Tseng, and Ming-Syan Chen. 2013. Highly Scalable Sequential Pattern Mining Based on MapReduce Model on the Cloud. In *IEEE International Congress on Big Data*. IEEE Computer Society, 310–317.
[19] 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.
[20] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. 2013. Discovering Denial Constraints. *PVLDB* (2013).
[21] Allan Peter Davis, Cynthia J. Grondin, Robin J. Johnson, Daniela Sciaky, Roy McMorran, Jolene Wiegers, Thomas C. Wiegers, and Carolyn J. Mattingly. 2019. The Comparative Toxicogenomics Database: update 2019. *Nucleic Acids Res.* 47, Database-Issue (2019), D948–D954.
[22] 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.
[23] J A. Dimasi. 2001. New drug development in the United States from 1963 to 1999. In *Clinical pharmacology and therapeutics vol. 69,5.*
[24] Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, and Denny Zhou. 2019. Neural Logic Machines. In *ICLR*.
[25] Joel T Dudley, Tarangini Deshpande, and Atul J Butte. 2011. Exploiting drug–disease relationships for computational drug repositioning. *Briefings in bioinformatics* 12, 4 (2011), 303–311.
[26] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq R. Joty, Mourad Ouzzani, and Nan Tang. 2018. Distributed Representations of Tuples for Entity Resolution. *PVLDB* (2018).
[27] Wenfei Fan, Hong Gao, Xibei Jia, Jianzhong Li, and Shuai Ma. 2011. Dynamic constraints for record matching. *VLDB J.* 20, 4 (2011), 495–520.
[28] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2008. Conditional functional dependencies for capturing data inconsistencies. *TODS* 33, 2 (2008), 6:1–6:48.
[29] Wenfei Fan, Floris Geerts, Jianzhong Li, and Ming Xiong. 2011. Discovering conditional functional dependencies. *TKDE* 23, 5 (2011), 683–698.
[30] Wenfei Fan, Floris Geerts, Nan Tang, and Wenyuan Yu. 2014. Conflict resolution with data currency and consistency. *J. Data and Information Quality* 5, 1-2 (2014), 6:1–6:37.
[31] Wenfei Fan, Ziyan Han, Yaoshu Wang, and Min Xie. 2022. Parallel Rule Discovery from Large Datasets by Sampling. In *SIGMOD*.
[32] Wenfei Fan, Ping Lu, and Chao Tian. 2020. Unifying Logic Rules and Machine Learning for Entity Enhancing. *Science China Information Sciences* (2020).
[33] Wenfei Fan, Chao Tian, Yanghao Wang, and Qiang Yin. 2021. Parallel Discrepancy Detection and Incremental Detection. *PVLDB* 14, 8 (2021), 1351–1364.
[34] Wenfei Fan, Xin Wang, Yinghui Wu, and Jingbo Xu. 2015. Association Rules with Graph Patterns. *PVLDB* 8, 12 (2015), 1502–1513.
[35] Peter A Flach and Iztok Savnik. 1999. Database dependency discovery: A machine learning approach. *AI communications* 12, 3 (1999), 139–160.
[36] Chris Fotis, Asier Antoranz, Dimitris Hatziavramidis, Theodore Sakellaropoulos, and Leonidas G. Alexopoulos. 2017. Pathway-based technologies for early drug discovery. *Drug Discovery Today* (2017).
[37] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. 2007. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics* 9, 3 (2007), 432–441.
[38] Cheng Fu, Xianpei Han, Le Sun, Bo Chen, Wei Zhang, Suhui Wu, and Hao Kong. 2019. End-to-end multi-perspective matching for entity resolution. In *IJCAI*. 4961–4967.
[39] Wensheng Gan, Jerry Chun-Wei Lin, Philippe Fournier-Viger, Han-Chieh Chao, and Philip S. Yu. 2019. A Survey of Parallel Sequential Pattern Mining. *ACM Trans. Knowl. Discov. Data* 13, 3 (2019), 25:1–25:34.
[40] Eve Garnaud, Nicolas Hanusse, Sofian Maabout, and Noël Novelli. 2014. Parallel mining of dependencies. In *HPCS*. IEEE, 491–498.
[41] Chang Ge, Ihab F. Ilyas, and Florian Kerschbaum. 2019. Secure Multi-Party Functional Dependency Discovery. *PVLDB* 13, 2 (2019), 184–196.
[42] Liqiang Geng and Howard J. Hamilton. 2006. Interestingness measures for data mining: A survey. *ACM Comput. Surv.* 38, 3 (2006), 9.
[43] Lise Getoor and Ashwin Machanavajjhala. 2012. Entity resolution: tutorial. *VLDB* (2012).
[44] Lukasz Golab, Howard Karloff, Flip Korn, Divesh Srivastava, and Bei Yu. 2008. On generating near-optimal tableaux for conditional functional dependencies. *VLDB* (2008).
[45] Valerie Guralnik and George Karypis. 2004. Parallel tree-projection-based sequence mining algorithms. *Parallel Comput.* 30, 4 (2004), 443–472.
[46] Jiawei Han, Jian Pei, and Yiwen Yin. 2000. Mining Frequent Patterns without Candidate Generation. In *SIGMOD*, Weidong Chen, Jeffrey F. Naughton, and Philip A. Bernstein (Eds.).
[47] Hado van Hasselt, Arthur Guez, and David Silver. 2016. Deep Reinforcement Learning with Double Q-Learning. In *AAAI* (Phoenix, Arizona) *(AAAI'16)*. 2094–2100.
[48] Alireza Heidari, Joshua McGrath, Ihab F. Ilyas, and Theodoros Rekatsinas. 2019. HoloDetect: Few-Shot Learning for Error Detection. In *SIGMOD*.
[49] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.
[50] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. 1999. TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies. *Comput. J.* (1999).
[51] Robert Kessl. 2021. Parallel algorithms for mining of frequent itemsets. *CoRR* abs/2108.05038 (2021).
[52] Hanna Köpcke, Andreas Thor, and Erhard Rahm. 2010. Evaluation of entity resolution approaches on real-world match problems. *PVLDB* 3, 1-2 (2010), 484–493.
[53] loannis Koumarelas, Thorsten Papenbrock, and Felix Naumann. 2020. MDedup: Duplicate detection with matching dependencies. *PVLDB* 13, 5 (2020), 712–725.
[54] Sebastian Kruse and Felix Naumann. 2018. Efficient discovery of approximate dependencies. *VLDB* (2018).
[55] Clyde P Kruskal, Larry Rudolph, and Marc Snir. 1990. A complexity theory of efficient parallel algorithms. *TCS* (1990).
[56] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942* (2019).
[57] Bing Li, Yukai Miao, Yaoshu Wang, Yifang Sun, and Wei Wang. 2021. Improving the Efficiency and Effectiveness for BERT-based Entity Resolution. In *AAAI*. AAAI Press, 13226–13233.
[58] Bing Li, Wei Wang, Yifang Sun, Linhan Zhang, Muhammad Asif Ali, and Yi Wang. 2020. GraphER: Token-Centric Entity Resolution with Graph Convolutional Neural Networks.. In *AAAI*. 8172–8179.
[59] Weibang Li, Zhanhuai Li, Qun Chen, Tao Jiang, and Hailong Liu. 2015. Discovering functional dependencies in vertically distributed big data. In *WISE*. 199–207.
[60] Weibang Li, Zhanhuai Li, Qun Chen, Tao Jiang, and Zhilei Yin. 2016. Discovering approximate functional dependencies from distributed big data. In *APWeb*. 289–301.
[61] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep Entity Matching with Pre-Trained Language Models. *arXiv preprint arXiv:2004.00584* (2020).
[62] Yen-Hui Liang and Shiow-Yang Wu. 2015. Sequence-Growth: A Scalable and Effective Frequent Itemset Mining Algorithm for Big Data Based on MapReduce Framework. In *2015 IEEE International Congress on Big Data*, Barbara Carminati and Latifur Khan (Eds.). IEEE Computer Society, 393–400.
[63] Martin S Lipsky and Lisa K Sharp. 2001. From idea to market: the drug approval process. *The Journal of the American Board of Family Practice* 14, 5 (2001),

362–367.
[64] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).
[65] Ester Livshits, Alireza Heidari, Ihab F. Ilyas, and Benny Kimelfeld. 2020. Approximate Denial Constraints. *PVLDB* 13, 10 (2020), 1682–1695.
[66] Stéphane Lopes, Jean-Marc Petit, and Lotfi Lakhal. 2000. Efficient discovery of functional dependencies and Armstrong relations. In *EDBT*. Springer, 350–364.
[67] Mohammad Mahdavi, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2019. Raha: A Configuration-Free Error Detection System. In *SIGMOD*. 865–882.
[68] Bundit Manaskasemsak, Nunnapus Benjamas, Arnon Rungsawang, Athasit Surarerks, and Putchong Uthayopas. 2007. Parallel association rule mining based on FI-growth algorithm. In *International Conference on Parallel and Distributed Systems, ICPADS*. IEEE Computer Society, 1–8.
[69] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nat.* (2015).
[70] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep learning for entity matching: A design space exploration. In *SIGMOD*.
[71] Noel Novelli and Rosine Cicchetti. 2001. Fun: An efficient algorithm for mining functional and embedded dependencies. In *ICDT*. Springer, 189–203.
[72] 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.
[73] Thorsten Papenbrock and Felix Naumann. 2016. A Hybrid Approach to Functional Dependency Discovery. In *SIGMOD*.
[74] 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.
[75] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proc. of NAACL*.
[76] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.
[77] Sanjay Rathee and Arti Kashyap. 2018. Adaptive-Miner: an efficient distributed association rule mining algorithm on Spark. *J. Big Data* 5 (2018), 6.
[78] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *EMNLP-IJCNLP*. 3980–3990.
[79] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *PVLDB* (2017).
[80] Theodoros Rekatsinas, Manas Joglekar, Hector Garcia-Molina, Aditya G. Parameswaran, and Christopher Ré. 2017. SLiMFast: Guaranteed Results for Data Fusion and Source Reliability. In *SIGMOD*. 1399–1414.
[81] 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.
[82] Hemant Saxena, Lukasz Golab, and Ihab F Ilyas. 2019. Distributed discovery of functional dependencies. In *ICDE*. IEEE, 1590–1593.
[83] Hemant Saxena, Lukasz Golab, and Ihab F Ilyas. 2019. Distributed implementations of dependency discovery algorithms. *PVLDB* 12, 11 (2019), 1624–1636.
[84] Philipp Schirmer, Thorsten Papenbrock, Ioannis Koumarelas, and Felix Naumann. 2020. Efficient Discovery of Matching Dependencies. *RODS* 45, 3 (2020), 1–33.
[85] 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*.
[86] Shaoyun Shi, Hanxiong Chen, Weizhi Ma, Jiaxin Mao, Min Zhang, and Yongfeng Zhang. 2020. Neural Logic Reasoning. In *CIKM*. 1365–1374.
[87] 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.
[88] Shaoxu Song and Lei Chen. 2009. Discovering matching dependencies. In *CIKM*.
[89] Ramakrishnan Srikant and Rakesh Agrawal. 1996. Mining Sequential Patterns: Generalizations and Performance Improvements. In *EDBT*, Peter M. G. Apers, Mokrane Bouzeghoub, and Georges Gardarin (Eds.). Springer, 3–17.
[90] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. ArnetMiner: Extraction and Mining of Academic Social Networks. In *KDD'08*. 990–998.
[91] Jeffrey D. Ullman. 2000. A Survey of Association-Rule Mining. In *International Conference on Discovery Science*, Setsuo Arikawa and Shinichi Morishita (Eds.). Springer, 1–14.
[92] Leslie G. Valiant. 1990. A Bridging Model for Parallel Computation. *Commun. ACM* 33, 8 (1990), 103–111.
[93] G. I. Webb and S. Zhang. 2005. k-Optimal Rule Discovery. *Data Mining and Knowledge Discovery* 10, 1 (2005), 39–79.
[94] Renzhi Wu, Sanya Chaba, Saurabh Sawlani, Xu Chu, and Saravanan Thirumuruganathan. 2020. ZeroER: Entity Resolution using Zero Labeled Examples. In *SIGMOD*. 1149–1164.
[95] 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*.
[96] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *NeurIPS*. 5753–5763.
[97] H Yao, H Hamilton, and C Butz. 2002. Fd_mine: discovering functional dependencies in a database using equivalences, Canada. In *IEEE ICDM*. 1–15.
[98] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. 2017. Deep Sets. In *Advances in Neural Information Processing Systems*. 3391–3401.
[99] Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. 1997. New Algorithms for Fast Discovery of Association Rules. In *KDD*, David Heckerman, Heikki Mannila, and Daryl Pregibon (Eds.). AAAI Press, 283–286.
[100] Xiangxiang Zeng, Xinqi Tu, Yuansheng Liu, Xiangzheng Fu, and Yansen Su. 2022. Toward better drug discovery with knowledge graph. *Current opinion in structural biology* 72 (2022), 114–126.
[101] Yunjia Zhang, Zhihan Guo, and Theodoros Rekatsinas. 2020. A Statistical Perspective on Discovering Functional Dependencies in Noisy Data. In *SIGMOD*. 861–876.
[102] Chen Zhao and Yeye He. 2019. Auto-EM: End-to-end Fuzzy Entity-Matching using Pre-trained Deep Models and Transfer Learning. In *WWW*. 2413–2424.