Discovering Top-k Rules using Subjective and Objective Criteria

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

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 134X on average.

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

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

ACM Reference Format:
Wenfei Fan, Ziyan Han, Yaoshu Wang, and Min Xie. 2023. Discovering Top-k Rules using Subjective and Objective Criteria. *Proc. ACM Manag. Data* 1, 1, Article 70 (May 2023), 29 pages. https://doi.org/10.1145/3588924

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 of 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 [123]. Drug discovery is costly and time-consuming, starting from target selection and validation, through preclinical screening, to clinical trials [41]. On average, the development of a new drug takes 15 years [27], costs $8M [7], and has a high risk of failure (>90% [12]). To shorten the 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

*Corresponding author

Authors' addresses: Wenfei Fan, Shenzhen Institute of Computing Sciences, China and University of Edinburgh, United Kingdom and Beihang University, China, wenfei@inf.ed.ac.uk; Ziyan Han, Beihang University, China, hanzy@act.buaa.edu.cn; Yaoshu Wang, Shenzhen Institute of Computing Sciences, China, yaoshuw@sics.ac.cn; Min Xie, Shenzhen Institute of Computing Sciences, China, xiemin@sics.ac.cn.

2836-6573/2023/5-ART70 $15.00
https://doi.org/10.1145/3588924

reduce false relations caused by noise, and for explanations to justify mined DDA and DDI [123].

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 dataset with 27 attributes and 368 tuples, 128,726 functional dependencies (FDs) are found [85]. Practitioners are overwhelmed by 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 useful "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, since excessive irrelevant rules could also be returned. What the domain experts want are to focus discovery on rules that fit their needs *e.g.,* rules that share vital characteristic with the rules they know effective and thus, help them identify more uses of drugs. 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 the most promising fraud patterns.

Example 1: To illustrate these needs, consider a pharmaceutical company who already knows that drug t_a can be used for disease s_a. What they want from rule discovery is to know whether t_a can be used to treat other diseases, *i.e.,* DDA for drug repurposing. The rule φ_1 may fit the need (note that φ_1 can be expressed by [24, 38, 72]).

φ_1 : Drug t_a is a potential treatment of disease s_b if (a) t_a has been used to cure disease s_a and (b) s_a and s_b have common therapies.

As recognized in [29], φ_1 is promising and should get a higher priority for clinical trials. In Section 2, we will formally express φ_1 and give more examples for other pharmaceutical needs, *e.g.,* for identifying the equivalence of generic and brand-name drugs [73]. □

This gives rise to several questions. How can we discover and rank rules to reflect their potential, and disclose more uses in addition to what the 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 data? Can we interpret discovered DDA and DDI, and cope with noise, which are critical but are still open in pharmaceutical scenarios [123]?

Contributions & organizations. This paper tackles these issues, exploring a new approach for discovering top-ranked rules. As proof of concept, we consider entity enhancing rules (REEs) [38], which were originally designed for ER and CR, and recently find new applications in association analysis. REEs (a) embed ML models in logic rules, (b) unify ER, CR and association analysis in the same process, (c) are collectively defined across multiple tables, and (d) subsume association rules [94], matching dependencies (MDs) [11, 13, 32], denial constraints (DCs) [10] and conditional functional dependencies (CFDs) [33, 35] 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 subjective measures in favor of rules that fit application needs, 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 [64], *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. (a) Top-k REEs discovery speeds up conventional methods by 134X on average. It takes 183s on NCVoter with 1,681,617 tuples for k = 10 with 20 machines, versus 15,798s by traditional methods. (b) The anytime algorithm is 52.8X faster than the top-k one when users want the 4th top-10 REEs. (c) Our bi-criteria model is 14.1% more accurate than the state-of-the-art language models, and its subjective measure improves the accuracy from 0.69 to 0.92. (d) The algorithms are parallelly scalable; they are 3.12X 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, to tackle the issues of excessive candidates and prohibitive cost in rule discovery, by incorporating human experience. It also provides the first top-k and anytime algorithms, with the parallel scalability, to demonstrate the effect of the bi-criteria model on finding top ranked rules and reducing the cost.

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 [57], FUN [84], FD_mine [120] discover FDs based on a lattice structure, which is latter extended by Depmine [77], HyFD [86] DynFD [104], and SMFD [47]. Levelwise methods for CFDs and MDs include CTANE [34], tableau generation [50] and [107]. [36] adopts sampling to discover REEs in large datasets. Apriori [8] and its variants (*e.g.,* DIC [17] and GSP [108]) employ breath-first search to mine association rules or frequent itemsets. (2) *Depth-first search.* DFD [5] and FastFDs [118] adopt the depth-first search in the lattice to mine FDs. For CFDs and DCs, depth-first search approaches also apply, *e.g.,* FastCFDs [34], FastDC [23], Hydra [15], DCFinder [87] and ADCMiner [76]. FP-growth [52] and Eclat [122] use depth-first search to mine association rules or frequent itemsets. (3) *Hybrid approaches.* HyMD [103] and MDedup [62] employ both levelwise and depth-first search to discover MDs. (4) *Learning-based approaches.* Inductive learning [40] and structure learning [124] have been utilized to find FDs. [106] and [61] adopt rule learning strategies to find MDs (see [22] for more ER solvers). Learned rules are also used for blocking and debugging, *e.g.,* Smurf [19]. Similar techniques also apply to association rule mining [99]. (5) *Top-k discovery.* Closer to this work are top-k algorithms for discovering association rules in relations [116] and graphs [39]. Mining of

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 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 colorless liquid rapidly absorbed from the gastrointestinal tract...	C_2H_6O	Miles lab 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 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

association rules and sequential patterns has also been studied in [45] and [111].

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, and 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 [48] for a survey). However, we are not aware of any rule discovery algorithms that take both measures into account. While MDedup [62] 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' application needs. While ML models [28, 105] 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-based approach to quantifying subjective measures. It differs from [39, 116] in the use of different ranking criteria and thus different early-termination strategies.

Rule learning. Rule learning has been studied for ER, CR and association analysis [98]. Rule induction methods such as IREP [44], RIPPERk [24] and TRIPPER [113] employ growing sets (resp. pruning sets) to learn rules (resp. reduce rules). Inductive logic programming (ILP) [83] utilizes automated reasoning and knowledge representation. Decision tree methods, *e.g.,* CART [16], ID3 [90], C4.5 [91] and C5.0 [92], apply greedy splitting strategies; its optimizations include GOSDT [72] for data imbalance and continuous variables, CORELS [9] via branch-and-bound algorithms with tight bounds, and OSDT [56] to construct optimal decision trees over binary variables. Bayesian rule set (BRS) [115] adopts a Bayesian method to build the rule set. Approximate inference [115], Monte-Carlo search [20], smart guessing [20] and modified prefix trees [9] have also been explored to prune the exponential search space.

This work differs from rule learning methods in that it aims to mine the top-k among *all* rules hold on the data, while rule learning methods aim to find some rules to minimize a predefined loss.

ML. ML models have been studied for ER, CR or association analysis. (1) *Models for ER and link prediction*, via logistic regression and SVM (see [49] for a survey), unsupervised learning, *e.g.,* ZeroER [117], and deep learning, *e.g.,* Ditto [70], BertER [66], DeepMatcher [82], DeepER [31], AutoEM [125], GraphER [67], MPM [43]. (2) *Models for CR, e.g.,* HoloClean [96], HoloDetect [54], Raha [78] and SLiMFast [97], for error detection/correction and data fusion. (3) *Models for similarity checking*, by using language models such as the BERT-based models [26, 65, 75, 95], XLNet [119] and GPT [18, 93].

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 2, 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 studied for mining FDs [46, 47, 68, 69, 101, 102], DCs [102], REEs [36], frequent itemsets [79], and sequence patterns [21, 51, 71] (see [45, 59] for surveys). This work provides the first top-k and anytime algorithms for relational data 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 [38].

We define REEs over a database schema $\mathcal{R} = (R_1, \ldots, R_m)$. Each R_j is a schema $R_j(A_1 : \tau_1, \ldots, A_n : \tau_n)$, where A_i is an attribute of type τ_i. An instance $\mathcal{D}$ of $\mathcal{R}$ is a collection $(D_1, \ldots, D_m)$, where D_i is a relation of R_i. We assume *w.l.o.g.* that each tuple t in $\mathcal{D}$ has an id attribute, which uniquely denotes the entity that t represents.

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 $\{=, \neq\}$. While REEs also support comparison operators, *i.e.,* $\{<, \leq, >, \geq\}$ [38], we consider $=$ and $\neq$ in this paper to simplify the discussion. Following tuple relational calculus [6], (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. We consider $\mathcal{M}$ such as (1) NLP models, *e.g.,* Bert [26], for text classification; (2) ER models and link prediction models, *e.g.,* ditto [70] and DeepMatcher [82], to reveal "hidden" associations between tuples across relations; and (3) models for data fusion, error detection and correction, *e.g.,* HoloClean [96] and HoloDetect [54].

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 2: 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, where the rule in Example 1 for identifying DDA can be written as φ_1 below.

(1) φ_1: $\text{Drug}(t_a) \wedge \text{Disease}(s_a) \wedge \text{Disease}(s_b) \wedge t_a.\text{cid} = s_a.\text{cid} \wedge \mathcal{M}_{\text{therapy}}(s_a, s_b) \rightarrow t_a.\text{cid} = s_b.\text{cid}$. Here $\mathcal{M}_{\text{therapy}}$ is an ML model for checking the common therapies of two diseases. This rule helps us discover the use of a known 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.

(2) $\varphi_2 : \text{Drug}(t_a) \wedge \text{Drug}(t_b) \wedge X \rightarrow \mathcal{M}_{\text{bio}}(t_a, t_b)$, 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 $\mathcal{M}_{\mathsf{bio}}$ 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 [73].

(3) φ_3 : $\mathsf{Drug}(t_a) \wedge \mathsf{Disease}(s_a) \wedge \mathsf{Disease}(s_b) \wedge t_a.\mathsf{cid} = s_a.\mathsf{cid} \wedge s_a.\mathsf{term} =$ "Atrophy" $\wedge$ $\mathcal{M}_{\mathsf{new}}(t_a, s_b) \wedge \mathcal{M}_{\mathsf{therapy}}(s_a, s_b) \rightarrow s_b.\mathsf{classification} =$ "Pathological", where $\mathcal{M}_{\mathsf{therapy}}$ is as above, and $\mathcal{M}_{\mathsf{new}}(t_a, s_b)$ 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 $\mathcal{M}_{\mathsf{therapy}}$), 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 $\mathcal{M}_{\mathsf{new}}$. This rules can be used to fix errors in the classification attribute. □

REEs embed ML classifiers in rules, unifying ER, CR and association analysis. With the comparison primitives, REEs strictly generalize common data quality rules (FDs, CFD, DCs and MDs) and association rules [38], and may carry multiple tuple variables (*e.g.,* 3 variables in φ_1) for collective analysis [14]. We can deduce DDA/DDI, give explanations and fix errors by REEs in a uniform framework.

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 [6]. (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 precondition X, 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 $\mathcal{D}$, $h \models \varphi$. We say that $\mathcal{D}$ *satisfies* a set Σ of REEs, denoted by $\mathcal{D} \models \Sigma$, if for all $\varphi \in \Sigma$, $\mathcal{D} \models \varphi$.

Example 3: Continuing with Example 2, assume that $\mathcal{D}$ has two relations D_1 and D_2 of schemas Drug and Disease, shown in Tables 1 and 2, respectively. Consider valuation h_1: $t_5 \mapsto t_a$, $t_{11} \mapsto s_a$ and $t_{12} \mapsto s_b$. This valuation satisfies φ_1 and discovers 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 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 users/applications.

3.1.1 Objective Measures. We start with objective measures.

Support. Support measures how frequently an REE can be applied. For collective rules across 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 4: Let X be $\mathsf{Drug}(t_a) \wedge t_a.\mathsf{name}$ = "Phenformin" and p_0 be $t_a.\mathsf{formula}$ = "$C_{10}H_{15}N_5$". Consider two REEs φ and φ', $\varphi : X \rightarrow p_0$ and $\varphi' : X' \rightarrow p_0$, where $X' = X \wedge \mathsf{Disease}(s_a) \wedge t_a.\mathsf{cid} = s_a.\mathsf{cid}$. 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. 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 other cases, *e.g.,* for 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 $\mathsf{spset}(p, \mathcal{D})$, as

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

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:

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

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 $\mathsf{spset}(\varphi, \mathcal{D})$, is defined as

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

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

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

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

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

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

Proof. There are two cases to consider: (1) φ and φ' use the same set of tuple variables. In this case, $\mathsf{spset}(\varphi', \mathcal{D})$ is clearly a subset of $\mathsf{spset}(\varphi, \mathcal{D})$, 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 $\mathsf{spset}(\varphi', \mathcal{D})$, $\langle h(t_0), h(s_0)\rangle$ must also be in $\mathsf{spset}(\varphi, \mathcal{D})$, since otherwise h cannot satisfy φ'. In both cases, $\mathsf{spset}(\varphi', \mathcal{D}) \subseteq \mathsf{spset}(\varphi, \mathcal{D})$ and thus $\mathsf{supp}(\varphi', \mathcal{D}) \leq \mathsf{supp}(\varphi, \mathcal{D})$. □

Example 5: In Example 4, $\varphi \preceq \varphi'$. By Theorem 1, $\mathsf{supp}(\varphi, \mathcal{D}) \geq \mathsf{supp}(\varphi', \mathcal{D})$, since $\mathsf{spset}(\varphi, \mathcal{D}) = \mathsf{spset}(\varphi', \mathcal{D}) = \{t_2 \mapsto t_a\}$. □

Confidence. Confidence indicates how often an REE $\varphi : X \rightarrow p_0$ has been found true, given that its precondition X is satisfied. For an REE $\varphi : X \rightarrow p_0$, the *confidence* of φ on $\mathcal{D}$, denoted by

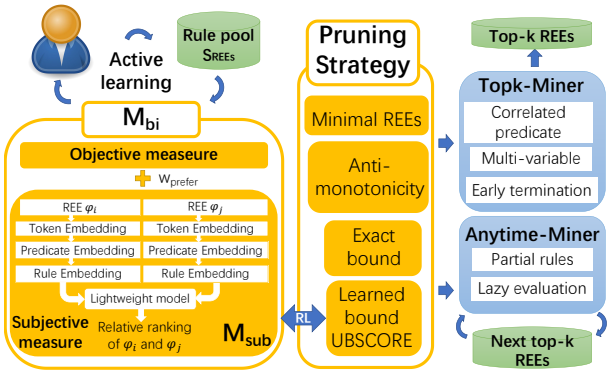


Fig. 1. Workflow

$\text{conf}(\varphi, \mathcal{D})$, is defined to be $\text{conf}(\varphi, \mathcal{D}) = \frac{|\text{spset}(X \wedge p_0, \mathcal{D})|}{|\text{spset}(X, \mathcal{D})|}$ (which is a value from 0 to 1).

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 φ' such that $\varphi' \preceq \varphi$.

Besides support and confidence, we can also use *attribute diversity* 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. Unlike "one-size-fit-all" objective measures, subjective measures could be used to capture individual users' needs or what a group of domain experts collectively think is valuable to an application. Below we train an ML model $\mathcal{M}_{\text{sub}}$ for catching subjective needs, 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 to a lightweight model, which outputs a scalar score, indicating the subjective preference of users. The details are explained as follows.

Assumption. We assume that meta-data (*e.g.,* schema and attribute names) are meaningful to a pre-trained model *e.g.,* ELMo [88] or Bert [26]. Otherwise, the performance of $\mathcal{M}_{\text{sub}}$ degrades since these models may fail to give accurate embeddings to ad-hoc tokens.

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 the 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 [88] or Bert [26], 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* [121] 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:

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

Lightweight model. Given embedding E_φ, our lightweight model is designed carefully, by employing a fully-connected layer along with a learnable parameter $\mathsf{UB}_{\mathsf{sub}}$ and the ReLU activation function:

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

where $\mathbf{w}_{\mathsf{light}} \in \mathbb{R}^{(d_r+d)\times 1}$ and $b_{\mathsf{light}} \in \mathbb{R}$ are parameters of the lightweight model. By adopting the ReLU activation function, $\mathsf{UB}_{\mathsf{sub}}$ can serve as a (loose) upper bound on the subjective score, *i.e.,* $\mathcal{M}_{\mathsf{sub}}(\varphi) \leq \mathsf{UB}_{\mathsf{sub}}$. Intuitively, this allows us to develop effective pruning strategies for early termination in top-k rule discovery.

Example 6: Consider φ_3 in Example 2. We first tokenize its predicates, *e.g.,* $\{\mathcal{M}_{\mathsf{new}}, t_a, s_b\}$ of $p = \mathcal{M}_{\mathsf{new}}(t_a, s_b)$, and transform them into embeddings using, *e.g.,* ELMo. The representation of p is generated using $\mathbf{w}_{\mathsf{emb}}$. 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 $\mathcal{M}_{\mathsf{sub}}$. □

Remark. Note that we do not adopt existing language models to acquire the rule-level representations for training $\mathcal{M}_{\mathsf{sub}}$, for the following reasons: (1) REEs do not follow natural language structure and thus, directly applying language model (*e.g.,* Bert [26]) 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 define the *ranking score* of an REE φ based on *linear scoring* functions, one of the most proliferate representations of user preference since the inception of utility modeling [30, 58]; as shown in [89], it can achieve a good trade-off among multiple criteria. Let F and G be the set of objective measures and subjective measures, respectively. The ranking score of φ is defined to be

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

where w_f and w_g are non-negative weights associated with the measures. Denote the complete weight vector by $\mathbf{w}_{\mathsf{prefer}}$, representing the user preference (*i.e.,* the individual need of a user or the application needs from a group of experts jointly). Intuitively, a larger weight indicates that the corresponding measure is more important. Once $\mathbf{w}_{\mathsf{prefer}}$ 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 $\mathbf{w}_{\mathsf{prefer}}$. Domain experts may want to find rules that fit the needs of their applications and thus, assign a larger weight to $\mathcal{M}_{\mathsf{sub}}$. A novice student 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. Note that a user does not need to give exact ranking scores for rules. What she/he needs to do is to conduct pairwise comparison on a few carefully selected rule pairs (see below), so that the weight vector $\mathbf{w}_{\mathsf{prefer}}$ and subjective measure $\mathcal{M}_{\mathsf{sub}}$ are learned implicitly, based on which the ranking of rules can be derived (see Exp-2 and the case study of Section 7 for how users may label rules).

As a real-life example, our drug-discovery users have accumulated quite a few rules for target

identification [123], and the conventional notions of support and confidence may produce excessive candidates, making it difficult to find useful rules. What they need is the subjective measure (*i.e.,* G); they set w_f small and let w_g dominate in $\mathbf{w}_{\text{prefer}}$. The subjective measure $\mathcal{M}_{\text{sub}}$ here suffices for finding rules that fit their needs, 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 $\mathbf{w}_{\text{prefer}}$. In other words, given an REE φ, $\mathcal{M}_{\text{bi}}$ outputs the ranking score $\text{score}(\varphi)$.

Intuitively, our design is inspired by the techniques from *Learning to Rank* [74]; 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. We aim to learn $\mathcal{M}_{\text{bi}}$ based on these partial orders of rules, so that $\mathcal{M}_{\text{bi}}$ can be used to rank both *seen* and *unseen* rules. Below we show how to jointly train $\mathcal{M}_{\text{bi}}$ via the *active learning* strategy.

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 she/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.

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

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

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 $\mathcal{S}_{\text{REEs}}$ that $\mathcal{M}_{\text{bi}}$ cannot distinguish well, *i.e.,* pairs that have the smallest differences in ranking scores, and ask users for labeling. To increase the diversity, we also randomly select a few rule pairs from S_{REEs} for users to label. 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 $\mathcal{S}_{\text{REEs}}$ without worrying about the rule validity (see [3]). We find that it often suffices for users to label 320 rule pairs in 5 rounds of interaction (160 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 consequence p_0, including all predicates correlated to p_0; we focus on mining REEs $\varphi\colon X \rightarrow p_0$ such that $X \subseteq \mathcal{P}_0$. We identify correlated attributes via graphical lasso [42] or LSTM [55].

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 $\mathcal{M}_{\text{bi}}$ for computing the ranking scores.

- *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 [87].

When the confidence threshold δ is set to 1, we focus discovery on *exact* rules (*i.e.,* no violation is allowed). However, in most settings, rules do not apply perfectly and thus, we set δ less than 1 to mine *approximate* rules. This may increase the number of rules returned and the cost of selecting desired rules. Nonetheless, with our bi-criteria model (esp. subjective measures), we can alleviate this via top-k discovery to prioritize those rules that truly fit the needs.

Workflow. As shown in Figure 1, we first train the bi-criteria model $\mathcal{M}_{\text{bi}}$ via active learning by interacting with the users using the rules in the pool $\mathcal{S}_{\text{REEs}}$. Given $\mathcal{M}_{\text{bi}}$, we discover top-k rules and next k rules using algorithms Topk-Miner and Anytime-Miner, respectively, by adopting pruning strategies for early termination. In particular, we employ an ML model UBSCORE, which takes the learned $\mathcal{M}_{\text{bi}}$ as input 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 discovered so far. Denote the k-th highest ranking score of rules in Σ by T_k. Assume that we are checking whether an 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 upper bound h_{ub} of $h(\varphi')$ is $h(\varphi)$ if $\varphi \preceq \varphi'$. (c) If a ranking measure does not have the above properties, it is referred to as *general*, *e.g.,* usually the subjective model, whose upper bound h_{ub} is UB_{sub}. The exact upper bound of score(φ) is the weighted sum of upper bounds h_{ub} of all measures (see [3] for details).

Example 7: Assume that $k = 3$ and the k-th highest ranking score T_k in Σ is 10 (see Figure 2), and we are currently processing φ. If the exact bound of the REEs expanded from φ is found to be 8, we stop the expansion of φ, since it will not yield any top-k rules. □

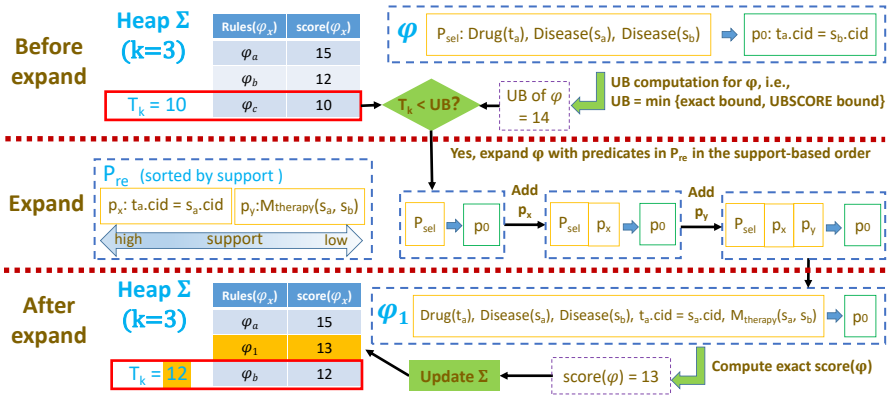


Fig. 2. A pictorial example of PTopk-Miner

(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 $\text{UBSCORE}(\varphi) \approx \max\{\mathcal{M}_{\text{sub}}(\varphi') \mid \varphi' : X \cup P' \rightarrow p_0, \forall P' \subseteq \mathcal{P}_0\}$. To learn UBSCORE, one may want to compute the exact φ' expanded from φ, with the maximum subjective score $\mathcal{M}_{\text{sub}}(\varphi')$ to get a training instance for UBSCORE (*i.e.,* use $\mathcal{M}_{\text{sub}}(\varphi')$ as its label). Then one can learn UBSCORE via a regression model (*e.g.,* a feed-forward network FFN, which is commonly used for predicting scalar values). However, computing the exact maximum subjective score $\mathcal{M}_{\text{sub}}$ is a combinatorial optimization problem with exponential enumeration cost.

As discussed in [60], a feasible way to tackle this issue is to learn a robust policy that heuristically constructs the training instances for UBSCORE in the reinforcement learning (RL) manner. Deep Q-learning (DQN) [81] is such an effective RL model for discrete action space and thus, we use DQN [81] to generate training instances of UBSCORE. Specifically, 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 [53] to speed up training.

After learning DQN, we generate N REEs as training instances for UBSCORE. We use the policy learned 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)

Algorithm Topk-Miner

Input: $\mathcal{R}$, $\mathcal{D}$, $\mathcal{P}_{\text{all}}$, k, σ, δ and $\mathcal{M}_{\text{bi}}$.
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) [87];
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. $\quad \Sigma := \text{Expand}(\mathcal{D}, \mathcal{P}_{\text{sel}}, \mathcal{P}_{\text{re}}, p_0, k, \delta, \sigma, \mathcal{M}_{\text{bi}}, \Sigma)$;
6. **return** Σ;

Procedure Expand

Input: $\mathcal{D}$, $\mathcal{P}_{\text{sel}}$, $\mathcal{P}_{\text{re}}$, p_0, k, δ, σ, $\mathcal{M}_{\text{bi}}$ 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** score(φ) $> \mathsf{T}_k$ **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 supp(φ) $< \sigma$ **[P2]** or 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;

Fig. 3. Algorithm Topk-Miner

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.

Example 8: Assume that we are expanding an REE $\varphi : X \rightarrow p_0$ with more predicates in $\mathcal{P}_0$ where $|\mathcal{P}_0| = 100$. Let $\varphi^* : X \wedge P^* \rightarrow p_0$ be the rule expanded from φ (*i.e.,* $P^* \subseteq \mathcal{P}_0$), with the maximum subjective score, *i.e.,* $\mathcal{M}_{\text{sub}}(\varphi^*) = \max_{\forall P' \subseteq \mathcal{P}_0}\{\mathcal{M}_{\text{sub}}(\varphi') \mid \varphi' : X \cup P' \rightarrow p_0\}$. Ideally, we want UBSCORE(φ) $\approx \mathcal{M}_{\text{sub}}(\varphi^*)$. Compared with a brute-force method which checks $O(2^{100})$ subsets to identify P^* from $\mathcal{P}_0$, if a robust heuristic policy is learned from DQN, P^* can be easily found, by iteratively selecting predicates from $\mathcal{P}_0$ with maximum rewards. After P^* is known, we can compute $\mathcal{M}_{\text{sub}}(\varphi^*)$ and get a training instance $(\varphi, \mathcal{M}_{\text{sub}}(\varphi^*))$ for UBSCORE. □

Remark. DQN is a general method for optimization and could be used in other rule mining algorithms, under the assumption that its interaction with environment is cheap. Intuitively, since DQN has to interact with the environment frequently for reward computation, its performance degrades if the interaction is costly. Nevertheless, in our setting, DQN only interacts with $\mathcal{M}_{\text{sub}}$, and it is fast.

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 3, 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 several optimization strategies. (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), *e.g.,* position list indexes (PLI) [87], to compute supports and confidences.

We also develop the following strategies for top-k REEs 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 [42, 124] 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 the set $\mathcal{P}_{\text{re}}$ 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 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 $\text{UB} = \min\{\text{exact bound}, \text{learned UBSCORE bound}\}$ is the score upper bound of the rules expanded from φ, then no rule expanded from φ has a higher score than those already in Σ; we can stop the expansion of φ immediately.

We further optimize Topk-Miner by considering two effective processing orders for $\mathcal{P}_{\text{re}}$ (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 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 9: 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 $\mathcal{P}_{\text{sel}}$:

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

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/score based processing orders can be combined, *e.g.,* sort by supports first and break ties by score indicators.

Example 10: We show how φ_1 in Example 2 is found (see Figure 2 for a pictorial illustration). Assume that $k = 3$, $\Sigma = \{\varphi_a, \varphi_b, \varphi_c\}$ with the k-th highest ranking score in Σ (*i.e.,* T_k) to be 10, $\mathcal{P}_{\text{sel}} = \{\text{Drug}(t_a), \text{Disease}(s_a), \text{Disease}(s_b)\}$, p_0 is t_a.cid = s_b.cid, and $\mathcal{P}_{\text{re}} = \{t_a.\text{cid} = s_a.\text{cid}, \mathcal{M}_{\text{therapy}}(s_a, s_b)\}$. We want to expand $\mathcal{P}_{\text{sel}}$ by adding predicates in $\mathcal{P}_{\text{re}}$, by the support-based processing order.

Before we perform the exact expansion, we first compute the upper bound, say UB. If $\text{UB} < \mathsf{T}_k$, we terminate early since expanding $\mathcal{P}_{\text{sel}}$ will not give any top-k REEs **[P3]**. Assume that UB is 14 and thus, $\mathsf{T}_k < \text{UB}$. Then t_a.cid = s_a.cid, which has a larger $\text{supp}(\mathcal{P}_{\text{sel}} \wedge p)$, is added to the precondition $\mathcal{P}_{\text{sel}}$ first, followed by the insertion of $\mathcal{M}_{\text{therapy}}(s_a, s_b)$, until we find that the resulting REE, say φ_1, is a minimal REE. If φ_1 has a higher score than some rules in Σ, Σ is updated accordingly, *e.g.,* in Figure 2, $\text{score}(\varphi_1) = 13 \geq \mathsf{T}_k$ and thus, Σ is updated by inserting φ_1, resulting in a tighter $\mathsf{T}_k = 12$. □

Complexity. Topk-Miner takes $O(\sum_{\varphi \in C(\mathcal{P}_0) \times \mathcal{P}_{\text{all}}} |\mathcal{D}|^{|\varphi|})$ time at worst, 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.

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 3) 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 the top-k rules, 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 *partial* if at the time it is processed, its score upper bound is lower than at least k complete rules in Σ. In other words, **[P3]** in 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 [37] 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 [38], we develop an efficient heuristic for checking, *i.e.,* an REE $\varphi : X \rightarrow p_0$ will not be added to Σ if there exists a set X' of predicates such that (1) $\varphi' : X' \rightarrow p_0$ is in Σ and (2) for each predicate p in X', $X \rightarrow p$ is also in Σ; intuitively, it means that p_0 can be "deduced" by some REEs that are already known and thus, is redundant.

Example 11: 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 [64].

Assume that $\mathcal{A}$ is a sequential algorithm which, given a dataset $\mathcal{D}$, a consequence 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:

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

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

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

6.2 Parallel Algorithm

We next parallelize Topk-Miner and develop PTopk-Miner (Figure 4). PTopk-Miner runs with one coordinator S_c and n workers $P_1, \ldots, P_n$ under the Bulk Synchronous Parallel (BSP) model [112], 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 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 $\mathcal{P}_{\mathsf{all}}$, S_c evenly divides it into n partitions, namely $\mathsf{RHS}_1, \ldots, \mathsf{RHS}_n$, and constructs a set of *work units* based on each RHS_j ($j \in [1, n]$) for the j-th worker P_j as follows.

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

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

Example 12: 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.

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

Parallel scalability. Below is the parallel scalability.

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

Proof. Recall the complexity of Topk-Miner is $t(|\mathcal{D}|, |\mathcal{P}_{\mathsf{all}}|, \sigma, \delta) = O(\sum_{\varphi \in C(\mathcal{P}_0) \times \mathcal{P}_{\mathsf{all}}} |\mathcal{D}|^{|\varphi|})$. We next show that the parallel runtime of PTopk-Miner is in $O(\frac{t(|\mathcal{D}|, |\mathcal{P}_{\mathsf{all}}|, \sigma, \delta)}{n})$. In PTopk-Miner, S_c conducts workload assignment and maintains the global top-k by collecting REEs from each worker. The former takes $O(|\mathcal{P}_{\mathsf{all}}|)$ 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

Algorithm PTopk-Miner

Input: $\mathcal{R}$, $\mathcal{D}$, $\mathcal{P}_{\text{all}}$, k, σ, δ, $\mathcal{M}_{\text{bi}}$, 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 $\mathcal{D}_{\mathcal{W}_j} = \{t \in \mathcal{D} \mid \exists s \in \mathcal{D}, p \in \mathcal{P}_0$ s.t. $h\langle t, s\rangle \models p$ 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 $\mathcal{W}_j$ **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 ($\text{score}(\varphi) > \mathsf{T}_k^i$);
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}, updates T_k^i to T_k^{i+1} and broadcasts T_k^{i+1} to all workers; $i := i + 1$;
16. **return** Σ_i;

Fig. 4. Algorithm PTopk-Miner

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, and (4) the effectiveness of PTopk-Miner.

Experimental setting. We start with the experimental setting [4].

Datasets. Following the setting in studies [76, 87], we used eight datasets, shown in Table 6. Adult, Airport, Hospital, 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) [15, 23] 10 times and then modifying their attributes using a program of [33].

ML models. We used three ML predicates in REEs: (a) the original SentenceBert [95] for checking semantic similarity between textual attributes where traditional logic predicates do not work well (*e.g.,* title and author in DBLP); (b) default ditto [70] for ER, where we generated training data following [110] and fine-tuned/trained it with 100 epochs; (c) a Bert-based model for assessing DDA (drug-disease association) in DrugDisease with labeled data [25].

To decide the best thresholds for ML models, we used the grid search strategy in the range [0, 1] with a step 0.05 [70] to evaluate model accuracy on validation data (10% of data), and select the one with maximum value. For the ease of presentation, we only report the threshold test for 3 models: (a) ditto for relation Author of DBLP, (b) SentenceBert for attribute name of Author, and

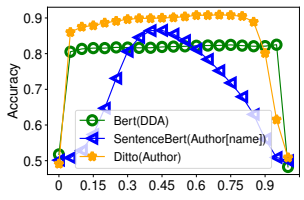


Fig. 5. Test ML thresholds

Name	Type	#tuples	#attributes	#relations
Adult [63, 76, 87]	real-life	32,561	15	1
Airport [76, 87]	real-life	55,113	18	1
Hospital [15, 23, 76, 87]	real-life	114,919	15	1
DrugDisease [25]	real-life	466,658	35	4
Inspection [76, 87, 96]	real-life	170,000	19	1
NCVoter [63, 76, 87]	real-life	1,681,617	12	1
DBLP [109]	real-life	1,799,559	18	3
Tax [15, 23, 33, 76, 87]	synthetic	10,000,000	15	1

Fig. 6. Dataset statistic

	Rule	supp	conf
φ_x	Hospital(t_0) ∧ Hospital(t_1) ∧ t_0.type = "Critical Access Hosplitals" ∧t_0.owner = "Proprietary" ∧t_1.state = "TX" ∧t_0.sample = t_1.sample ∧ t_0.stat_avg = t_1.stat_avg → t_0.emergency_service = **"No"**	1749	1.0
φ_y	Hospital(t_0) ∧ Hospital(t_1) ∧ t_1.owner = "Government Local" ∧t_0.state = t_1.state ∧ t_0.country = t_1.country ∧ t_0.sample = t_1.sample → t_0.zip = t_1.zip	8038	0.938

Table 3. A sample rule pair for comparison

(c) Bert for DDA in Figure 5; other models are similar. Take SentenceBert for author names as an example. As the threshold δ increases, the accuracy gets larger and then reaches the peak at $\delta = 0.45$. Thus we set 0.45 as the threshold for this SentenceBert; similarly for other models.

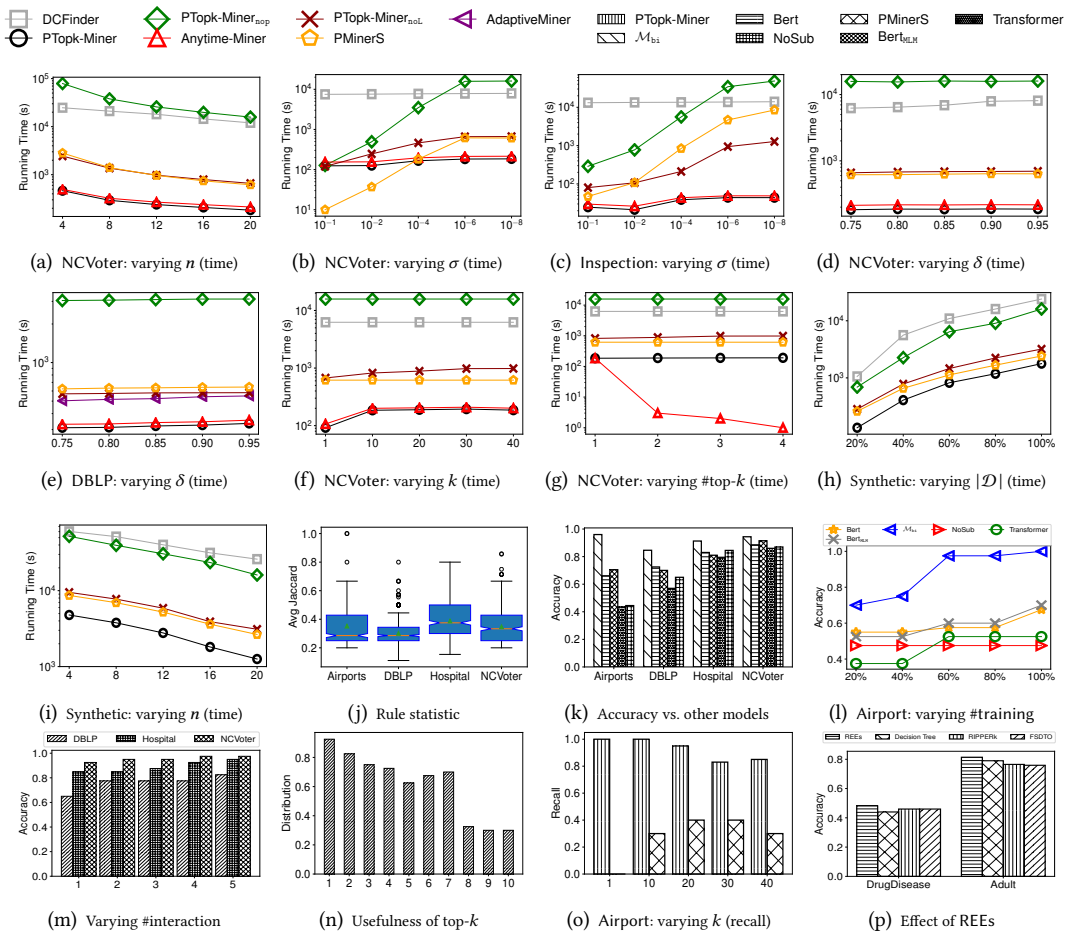
For our bi-criteria model $\mathcal{M}_{\text{bi}}$, we used the DistilBert [100] to initialize token embeddings and set the size of rule embeddings to 100. Token embeddings are also learned together during training. For UBSCORE, we adopted FFN with three 200-dimensional hidden layers. We used Adam optimizer to train $\mathcal{M}_{\text{bi}}$ (resp. UBSCORE) with a batch-size of 128 (resp. 64), and the learning rate is 10^{-3} (resp. 10^{-4}). We adopted 300 epochs on Tesla V100 GPU. The inferences of $\mathcal{M}_{\text{bi}}$ and UBSCORE are re-implemented using the EJML library [2].

Baselines. We implemented the following for top-k discovery, all in Java: (1) PTopk-Miner. (2) Anytime-Miner. (3) PTopk-Miner$_{\text{nop}}$, a variant of PTopk-Miner that mines all rules, sorts them by ranking scores, and returns the top-k ones. (4) PTopk-Miner$_{\text{noL}}$, another variant without the learned bound UBSCORE. (5) Topk-Filter, a variant of PTopk-Miner$_{\text{nop}}$ that ditches objective measures from ranking and uses them only for filtering, *i.e.,* it mines all rules, filters those with low objective scores and ranks the remaining with only subjective scores. (6) DCFinder [87], a DC discovery method which is shown to outperform other DCs methods [87]; we parallelize it by [102] and extend it to support constant and ML predicates by adding them into the evidence set [87]. (7) PMinerS [36], a parallel REE discovery method by sampling. (8) AdaptiveMiner [94], a MapReduce-based algorithm for mining association rules.

We compared PTopk-Miner$_{\text{nop}}$ and PTopk-Miner$_{\text{noL}}$ 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. Here PTopk-Miner$_{\text{nop}}$, Topk-Filter, DCFinder, AdaptiveMiner and PMinerS perform *full discovery*, *i.e.,* they have to mine all rules from the data (entire or sampled), sort (or filter) by ranking scores and return the top-k, while PTopk-Miner and Anytime-Miner incorporate $\mathcal{M}_{\text{bi}}$ into the discovery process explicitly to achieve speed-up. Unless stated explicitly, all baselines used the same bi-criteria model $\mathcal{M}_{\text{bi}}$ to compute ranking scores.

Moreover, we implemented the following to evaluate our bi-criteria model $\mathcal{M}_{\text{bi}}$: (1) Bert [26], a state-of-the-art language model (the distilbert-base-uncased), as a binary classifier with rule pairs as input, separated by [SEP]; a softmax layer is added after the embedding of [CLS]. (2) Bert$_{\text{MLM}}$, a variant of Bert that is further pre-trained on rules with masked-language modeling (MLM). (3) Transformer [114], an encoder-decoder network via multi-head self-attention. (4) NoSub, a variant of $\mathcal{M}_{\text{bi}}$ without subjective measures.

We conducted experiments on a cluster of up to 21 virtual machines (one for the coordinator), each



(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) NCVoter: varying #top-k (time) (h) Synthetic: varying $|\mathcal{D}|$ (time)

(i) Synthetic: varying n (time) (j) Rule statistic (k) Accuracy vs. other models (l) Airport: varying #training

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

Fig. 7. Performance evaluation

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 $\mathcal{M}_{\text{bi}}$ was trained via active learning once offline.

Experimental results. We next report our findings.

Exp-1: Scalability test. We first evaluated the scalability of PTopk-Miner and Anytime-Miner. As will be seen shortly, our top-k discovery method is effective; with the pruning strategies, it can achieve an 134X speedup on average. Unless stated explicitly, the default setting in our experiments is $n = 20$, $\sigma = 10^{-6} \cdot |\mathcal{D}|^2$, $\delta = 0.75$ and $k = 10$. 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. As shown in Figure 7(a), (a) PTopk-Miner scales well with the increase of machines: it is 3.15X 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 99.40X and 4.29X faster than PTopk-Miner$_{\text{nop}}$ and PTopk-Miner$_{\text{noL}}$ on average, up to 110.65X and 5.25X, respectively. This verifies the effectiveness of our pruning strategies and the learned bound UBSCORE. We omitted Topk-Filter

since it also mines all REEs and is thus as slow as PTopk-Miner$_{\text{nop}}$. (d) PTopk-Miner is 67.24X faster than DCFinder on average, up to 75.80X, showing the advantage of top-k discovery over traditional full discovery, even though it discovers more expressive REEs. (e) Although PMinerS runs on small samples, it is 4.37X slower than PTopk-Miner on average, which further verifies the effectiveness of the pruning strategies. We will discuss its accuracy in Exp-3. (e) Anytime-Miner is slightly slower than PTopk-Miner, since it maintains more rules in the heap (Section 5). Nonetheless, its advantage is evident when users continuously want next top-k rules (see below).

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 7(b) and 7(c). As expected, all algorithms take longer when σ is smaller since they examine more candidates, *e.g.,* PTopk-Miner$_{\text{nop}}$ is 147.57X 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 Figure 7(a). While PMinerS is slightly faster than PTopk-Miner for large σ on NCVoter, it sacrifices the accuracy, which will be discussed in Exp-3. 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 7(d), (a) most algorithms are faster given a smaller δ, *e.g.,* PTopk-Miner and Anytime-Miner are 1.02X and 1.11X 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.

Note that PTopk-Miner also performs the best for association rules (essentially constant CFDs). Figure 7(e) shows the discovery time to mine such rules on DBLP, when we set $\sigma = 10^{-4}|\mathcal{D}|$. PTopk-Miner is 1.64X faster than AdaptiveMiner on average.

Varying k. As shown in Figure 7(f) by varying k from 1 to 40, full discovery variants PTopk-Miner$_{\text{nop}}$, PMinerS and DCFinder are indifferent to k, since they mine all rules (*i.e.,* REEs or DCs) from the dataset (entire or sampled) regardless of k. In contrast, PTopk-Miner takes much less time due to its pruning strategies for top-k discovery; on average, it is 101.41X, 3.96X and 40.05X faster than the three, respectively. This again justifies the use of top-k discovery over full discovery. Anytime-Miner is also faster than the three, by adopting lazy evaluation for skipping unnecessary REEs expansions. The costs of PTopk-Miner and PTopk-Miner$_{\text{noL}}$ increase when k gets larger since more REEs have to be checked, as expected.

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 Figure 7(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 95.0X faster then PTopk-Miner. 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 4th top-10 on NCVoter, its runtime may decrease rapidly because most of rules in next #top-k are already 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 NCVoter.

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 7(h), all algorithms take longer, as expected. PTopk-Miner still outperforms all competitors; for instance, it takes PTopk-Miner 0.5h to run 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 7(i). Consistent with Figures 7(a). PTopk-Miner is 3.60X faster when n varies from 4 to 20.

Exp-2: Effectiveness of the bi-criteria model. In this experiment, domain experts (our industry partners), who well understood the requirements of their applications, were *independently* invited to label 400 pairs of distinct rules. Rules used for training were generated by combining predicates without worrying the validity (Section 3.2), on 1-3 relations, with an average of 5.45 predicates (see Table 3 for a sample rule pair displayed to the experts, to be explained in Exp-4). To better visualize the distribution, we compute the Jaccard similarity between each pair of rules and plot them in boxplot in Figure 7(j). As shown there, both diverse and similar rule pairs are in the rule pool, to optimize the information gain.

For each pair $\langle \varphi_i, \varphi_j \rangle$ of rules, the expert labeled 1 on $\langle \varphi_i, \varphi_j \rangle$ if she/he ranked φ_i higher than φ_j; otherwise, she/he labeled 0. To facilitate decision making, the experts were also provided with the objective measures and data instance samples that satisfy these rules (not shown in Table 3). For each expert, we trained a bi-criteria model by splitting her/his labeled pairs into training, validation and testing sets as 80%, 10% and 10%, respectively; the accuracy in the testing data was measured by the percentage of rule pairs whose relative rank is correctly identified. For a fair comparison, $\mathcal{M}_{\text{bi}}$ and all baselines were trained/fine-tuned on the same sets of labeled rule pairs. Below we report the average accuracy of the models from all experts. One can also train an "aggreated" model by aggregating opinions from all experts; its performance is similar (not shown).

Accuracy vs. language models. We first compared $\mathcal{M}_{\text{bi}}$ against Bert, Bert$_{\text{MLM}}$ and Transformer. As shown in Figure 7(k), it consistently beats the three in accuracy, *e.g.,* on DBLP, our accuracy is 0.846, which is 12%, 14% and 27% higher than the three, respectively.

We then varied the size of training data in Figure 7(l); all methods are more accurate given more training data, *e.g.,* the accuracy of Bert on Airport increases from 0.55 to 0.675, with 80% more training data. In almost all cases, $\mathcal{M}_{\text{bi}}$ still has the best accuracy, since (a) the unique feature, *i.e.,* permutation invariant, of logic rules cannot be captured by existing language models that learn rules as natural language, (b) a language model typically has millions of parameters and often over-fits with small data, and (c) the embedding domains of the pre-training of language models (*e.g.,* large corpus) and the downstream task (*i.e.,* rule discovery) are different.

Effectiveness of subjective measures. We next studied the effectiveness of subjective measures by comparing the accuracy of our model $\mathcal{M}_{\text{bi}}$ against NoSub. The results in Figure 7(k) verify that objective measures alone do not suffice since users can have diverse application needs; without the subjective measures, NoSub is reduced to linear regression and thus, is not accurate, *e.g.,* on DBLP, introducing subjective measures improves the accuracy by around 20%.

Varying #interaction. To verify the usefulness of active learning, we report the accuracy by varying the number #r of rounds of interaction in Figure 7(m). The experts labeled 40 pairs of rules in each round after 160 initial ones, a workload acceptable to domain experts. With larger #r, the accuracy increases, *e.g.,* after 5 rounds, accuracy changes from 0.85 to 0.95 on Hospital. 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 reach a stable accuracy, and the improvement is substantial (*e.g.,* 17% on DBLP) with only 160 extra labeled instances.

Usefulness of top-k REEs. To evaluate whether we rank rules reasonably, we set up 10 *independent* tasks over five datasets (Airport, DBLP, Inspection, NCVoter, and Adult); on each dataset, we mined rules using two consequence sets. Note that each task has its own purpose, *e.g.,* Adult with consequence t.class = "$\leq$ 50K" aims to find rules for identifying the factors of low salary. Below we report the average performance of top-k REEs over these 10 tasks. For each task, we used the 400 rule pairs labeled by an expert P to train $\mathcal{M}_{\text{bi}}$. Given the objective/subjective measures learned, we mine two lists of top-10 REEs using (1) PTopk-Miner and (2) Topk-Filter, which keep 100 REEs with maximum objective scores and ranks the final top-10 by the subjective scores. The 20 rules (*i.e.,* 10 from each list) are shuffled and presented to another expert Q different from P (the one who trained the model), who is instructed with the same task purpose as P, to label each of these rules (1 if a rule is useful).

We compute the number of "votes" (*i.e.,* labels) received by each rule and plot the percentage of "votes" for each of top-10 REEs returned by PTopk-Miner in Figure 7(n). It shows that REEs ranked higher are more favored, *e.g.,* REEs ranked first to fifth are labeled as useful by 77% of users on average, as opposed to 46% for REEs ranked sixth to tenth. The user ranking justifies the semantic of top-k discovery. We also compared the "votes" received by PTopk-Miner and Topk-Filter; on average, 6.15 REEs of PTopk-Miner are labeled as useful, as opposed to 5.1 by Topk-Filter. This verifies that our design flexibly combines both measures to suit different needs.

Exp-3: Accuracy test. We evaluated the accuracy of top-k discovery. Note that PTopk-Miner$_{\text{nop}}$ did not use any pruning strategies and thus it returns exact top-k; we compare it to evaluate the impact of the learned bound UBSCORE on the accuracy. 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 Figure 7(o). Recall is defined as $\frac{|\Sigma_{\text{M}} \cap \Sigma_{\text{GT}}|}{k}$, 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$_{\text{nop}}$). PTopk-Miner$_{\text{noL}}$ returns the exact top-k as PTopk-Miner$_{\text{nop}}$ (not shown), since the exact bound does not miss any rules. The recall of PTopk-Miner is close to PTopk-Miner$_{\text{nop}}$. PMinerS is not accurate, *e.g.,* 0.3, as opposed to 1.0 by PTopk-Miner when $k = 10$. We omit precision since it is the same as recall in this setting.

Exp-4: Effectiveness of PTopk-Miner**.** Finally, we present the case study of the effectiveness.

Effectiveness. We evaluated the accuracy of rules from PTopk-Miner vs. three baselines (decision tree (DT), RIPPERk [24], FSDTO [80]) on labeled data DrugDisease and Adult, for predicting half-life time of drugs (multi-classification) and incomes (binary classification), respectively. The baselines learn rules, from which top-k rules are ranked via our bi-criteria model. For drugs, we transform scalar values of attribute half_life_hours_curated to 4 categories. We split data for training/validation/testing with 80%/10%/10%. As shown in Figure 7(p), PTopk-Miner is is on average 3.3%, 3.52%, 3.82% more accurate than the three, up to 4.1%, 4.90% and 5.5%, respectively.

Case study. In the following, we (a) showcase a few real REEs discovered by PTopk-Miner on DrugDisease and Inspection with $\sigma \geq 10$ and $\delta \geq 0.85$. and (b) show how to label a sample rule pair.

(1) Rule 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) Rule 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 = $(-4, 1)$, C2 = $(0, 1)$, C3 = $(250, 350)$, C4 = $(0, 15)$ are intervals, and cal_logp, cal_water_solubility, cal_molecular_weight are features of drugs (defined in [1]). This rule says that if the features fit the ranges, then it takes <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 factors of short half-life time.

(3) An REE on Inspection ($\sigma \geq 10$ and $\delta \geq 0.85$): 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. It says two firms are equally risky if they have the same names, one with inspection type Canvass and the other with inspection result Pass. It ranks high for inspectors who assess the risk level of regulated facilities to determine a firm's compliance with laws and regulations.

(4) Consider a sample rule pair $\langle \varphi_x, \varphi_y \rangle$ displayed to the experts for labeling in Table 3; both are mined from Hospital. Here φ_x checks whether a hospital provides emergency service, by considering its type, owner and location etc.; it is reliable since conf(φ_x) = 1; φ_y describes a more common issue of deducing the zipcode of a hospital by its state and country, as reflected by its higher support than φ_x. If a user wants to know whether a hospital has emergency service, she/he can label 1 on $\langle \varphi_x, \varphi_y \rangle$, indicating that φ_x better fits the need.

Summary. We find the following. (1) Top-k discovery speeds up PTopk-Miner$_{\text{nop}}$, PTopk-Miner$_{\text{noL}}$, PMinerS and DCFinder by 134X, 6X, 14X and 86X on average over all settings, respectively, up to 168X, 12X, 65X and 138X. 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, 611s and 12,003s by the four. (2) PTopk-Miner also performs better than AdaptiveMiner for mining association rules. (3) PTopk-Miner scales well with parameters σ, δ and k. (4) PTopk-Miner is parallelly scalable: on average, it is 3.12X faster when the number n of machines varies from 4 to 20. (5) The lazy evaluation strategy of Anytime-Miner is effective: Anytime-Miner is 52.8X faster than PTopk-Miner when the users want the 4th top-10 REEs. (6) Our pruning strategies and the learned bound UBSCORE are effective, *e.g.,* reducing the the runtime of PTopk-Miner by 12.79X and 5.19X on the Tax data of 10M tuples. (7) Our bi-criteria model is on average 14.1% more accurate than language models. The subjective measure improves the accuracy from 0.69 to 0.92. (8) PTopk-Miner are able to find useful REEs beyond DCs by DCFinder and association rules by AdaptiveMiner, which are 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 future topic is to study incremental top-k discovery in response to data updates. Another topic is to integrate top-k discovery and sampling, to speed up discovery with accuracy guarantees.

ACKNOWLEDGMENTS

This work was supported by Royal Society Wolfson Research Merit Award WRM/R1/180014, Guangdong Basic and Applied Basic Research Foundation 2022A1515010120 and NSFC 62202313.

REFERENCES

[1] 2022. Drugbank. https://docs.drugbank.com/xml/#external-codes.
[2] 2022. EJML library. http://ejml.org/wiki/index.php?title=Main_Page.
[3] 2022. Full version. https://www.dropbox.com/sh/43p08bskxef96m2/AADjihrBKVxvgPXwkVcRKIHla?dl=0&preview=paper-full-version.pdf.
[4] 2022. Source code. https://www.dropbox.com/sh/43p08bskxef96m2/AADjihrBKVxvgPXwkVcRKIHla?dl=0.
[5] Ziawasch Abedjan, Patrick Schulze, and Felix Naumann. 2014. DFD: Efficient functional dependency discovery. In *CIKM*. 949–958.
[6] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases*. Addison-Wesley.
[7] 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.
[8] 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.).
[9] Elaine Angelino, Nicholas Larus-Stone, Daniel Alabi, Margo I. Seltzer, and Cynthia Rudin. 2017. Learning Certifiably Optimal Rule Lists for Categorical Data. *J. Mach. Learn. Res.* 18 (2017), 234:1–234:78.
[10] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. 1999. Consistent Query Answers in Inconsistent Databases. In *PODS*.
[11] Zeinab Bahmani and Leopoldo E. Bertossi. 2017. Enforcing Relational Matching Dependencies with Datalog for Entity Resolution. In *FLAIRS*.
[12] Nurken Berdigaliyev and Mohamad Aljofan. 2020. An overview of drug discovery and development. *Future Medicinal Chemistry* 12, 10 (2020), 939–947.
[13] 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).
[14] Indrajit Bhattacharya and Lise Getoor. 2007. Collective entity resolution in relational data. *TKDD* (2007).
[15] Tobias Bleifuß, Sebastian Kruse, and Felix Naumann. 2017. Efficient Denial Constraint Discovery with Hydra. *PVLDB* 11, 3 (2017), 311–323.
[16] Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. 1984. *Classification and Regression Trees*. Wadsworth.
[17] 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.
[18] 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).
[19] 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.
[20] Chaofan Chen and Cynthia Rudin. 2018. An Optimization Approach to Learning Falling Rule Lists. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, Vol. 84. PMLR, 604–612.
[21] Chun-Chieh Chen, Chi-Yao Tseng, and Ming-Syan Chen. 2013. Highly Scalable Sequential Pattern Mining Based on MapReduce Model on the Cloud. In *International Congress on Big Data*. IEEE, 310–317.
[22] 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.
[23] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. 2013. Discovering Denial Constraints. *PVLDB* 6, 13 (2013), 1498–1509.
[24] William W. Cohen. 1995. Fast Effective Rule Induction. In *International Conference on Machine Learning*. Morgan Kaufmann, 115–123.
[25] 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.
[26] 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.
[27] J A. Dimasi. 2001. New drug development in the United States from 1963 to 1999. In *Clinical pharmacology and therapeutics vol. 69,5*.
[28] Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, and Denny Zhou. 2019. Neural Logic Machines. In *ICLR*.
[29] 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.
[30] James Dyer and Rakesh Sarin. 1979. Measurable Multiattribute Value Functions. *Operations Research* 27 (08 1979), 810–822.
[31] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq R. Joty, Mourad Ouzzani, and Nan Tang. 2018. Dis-

tributed Representations of Tuples for Entity Resolution. *PVLDB* 11, 11 (2018), 1454–1467.
[32] Wenfei Fan, Hong Gao, Xibei Jia, Jianzhong Li, and Shuai Ma. 2011. Dynamic constraints for record matching. *VLDB J.* 20, 4 (2011), 495–520.
[33] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2008. Conditional functional dependencies for capturing data inconsistencies. *TODS* 33, 2 (2008), 6:1–6:48.
[34] Wenfei Fan, Floris Geerts, Jianzhong Li, and Ming Xiong. 2011. Discovering conditional functional dependencies. *TKDE* 23, 5 (2011), 683–698.
[35] 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.
[36] Wenfei Fan, Ziyan Han, Yaoshu Wang, and Min Xie. 2022. Parallel Rule Discovery from Large Datasets by Sampling. In *SIGMOD*.
[37] Wenfei Fan, Ping Lu, and Chao Tian. 2020. Unifying Logic Rules and Machine Learning for Entity Enhancing. *Science China Information Sciences* (2020).
[38] Wenfei Fan, Chao Tian, Yanghao Wang, and Qiang Yin. 2021. Parallel Discrepancy Detection and Incremental Detection. *PVLDB* 14, 8 (2021), 1351–1364.
[39] Wenfei Fan, Xin Wang, Yinghui Wu, and Jingbo Xu. 2015. Association Rules with Graph Patterns. *PVLDB* 8, 12 (2015), 1502–1513.
[40] Peter A Flach and Iztok Savnik. 1999. Database dependency discovery: A machine learning approach. *AI communications* 12, 3 (1999), 139–160.
[41] Chris Fotis, Asier Antoranz, Dimitris Hatziavramidis, Theodore Sakellaropoulos, and Leonidas G. Alexopoulos. 2017. Pathway-based technologies for early drug discovery. *Drug Discovery Today* (2017).
[42] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. 2007. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics* 9, 3 (2007), 432–441.
[43] 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.
[44] Johannes Fürnkranz and Gerhard Widmer. 1994. Incremental Reduced Error Pruning. In *International Conference on Machine Learning*. Morgan Kaufmann, 70–77.
[45] 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.
[46] Eve Garnaud, Nicolas Hanusse, Sofian Maabout, and Noël Novelli. 2014. Parallel mining of dependencies. In *HPCS*. IEEE, 491–498.
[47] Chang Ge, Ihab F. Ilyas, and Florian Kerschbaum. 2019. Secure Multi-Party Functional Dependency Discovery. *PVLDB* 13, 2 (2019), 184–196.
[48] Liqiang Geng and Howard J. Hamilton. 2006. Interestingness measures for data mining: A survey. *ACM Comput. Surv.* 38, 3 (2006), 9.
[49] Lise Getoor and Ashwin Machanavajjhala. 2012. Entity resolution: Tutorial. *PVLDB* (2012).
[50] Lukasz Golab, Howard Karloff, Flip Korn, Divesh Srivastava, and Bei Yu. 2008. On generating near-optimal tableaux for conditional functional dependencies. *VLDB* (2008).
[51] Valerie Guralnik and George Karypis. 2004. Parallel tree-projection-based sequence mining algorithms. *Parallel Comput.* 30, 4 (2004), 443–472.
[52] Jiawei Han, Jian Pei, and Yiwen Yin. 2000. Mining Frequent Patterns without Candidate Generation. In *SIGMOD*.
[53] Hado van Hasselt, Arthur Guez, and David Silver. 2016. Deep Reinforcement Learning with Double Q-Learning. In *AAAI* (Phoenix, Arizona). 2094–2100.
[54] Alireza Heidari, Joshua McGrath, Ihab F. Ilyas, and Theodoros Rekatsinas. 2019. HoloDetect: Few-Shot Learning for Error Detection. In *SIGMOD*.
[55] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.
[56] Xiyang Hu, Cynthia Rudin, and Margo I. Seltzer. 2019. Optimal Sparse Decision Trees. In *NeurIPS*. 7265–7273.
[57] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. 1999. TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies. *Comput. J.* (1999).
[58] Ralph Keeney, Howard Raiffa, and David Rajala. 1979. Decisions with Multiple Objectives: Preferences and Value Trade-Offs. *Systems, Man and Cybernetics, IEEE Transactions on* 9 (08 1979), 403 – 403.
[59] Robert Kessl. 2021. Parallel algorithms for mining of frequent itemsets. *CoRR* abs/2108.05038 (2021).
[60] Elias B. Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. 2017. Learning Combinatorial Optimization Algorithms over Graphs. In *NeurIPS*.
[61] 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.
[62] Ioannis Koumarelas, Thorsten Papenbrock, and Felix Naumann. 2020. MDedup: Duplicate detection with matching

dependencies. *PVLDB* 13, 5 (2020), 712–725.
[63] Sebastian Kruse and Felix Naumann. 2018. Efficient discovery of approximate dependencies. *PVLDB* 11, 7 (2018), 759–772.
[64] Clyde P Kruskal, Larry Rudolph, and Marc Snir. 1990. A complexity theory of efficient parallel algorithms. *TCS* 71, 1 (1990), 95–132.
[65] 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).
[66] 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.
[67] 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.
[68] Weibang Li, Zhanhuai Li, Qun Chen, Tao Jiang, and Hailong Liu. 2015. Discovering functional dependencies in vertically distributed big data. In *WISE*. 199–207.
[69] Weibang Li, Zhanhuai Li, Qun Chen, Tao Jiang, and Zhilei Yin. 2016. Discovering approximate functional dependencies from distributed big data. In *APWeb*. 289–301.
[70] 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).
[71] 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 *International Congress on Big Data*. IEEE, 393–400.
[72] Jimmy Lin, Chudi Zhong, Diane Hu, Cynthia Rudin, and Margo I. Seltzer. 2020. Generalized and Scalable Optimal Sparse Decision Trees. In *International Conference on Machine Learning (ICML)*, Vol. 119. PMLR, 6150–6160.
[73] 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.
[74] Tie-Yan Liu. 2009. Learning to Rank for Information Retrieval. *Found. Trends Inf. Retr.* 3, 3 (2009), 225–331. https://doi.org/10.1561/1500000016
[75] 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).
[76] Ester Livshits, Alireza Heidari, Ihab F. Ilyas, and Benny Kimelfeld. 2020. Approximate Denial Constraints. *PVLDB* 13, 10 (2020), 1682–1695.
[77] Stéphane Lopes, Jean-Marc Petit, and Lotfi Lakhal. 2000. Efficient discovery of functional dependencies and Armstrong relations. In *EDBT*. Springer, 350–364.
[78] 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.
[79] Bundit Manaskasemsak, Nunnapus Benjamas, Arnon Rungsawang, Athasit Surarerks, and Putchong Uthayopas. 2007. Parallel association rule mining based on FI-growth algorithm. In *ICPADS*. IEEE, 1–8.
[80] Hayden McTavish, Chudi Zhong, Reto Achermann, Ilias Karimalis, Jacques Chen, Cynthia Rudin, and Margo I. Seltzer. 2022. Fast Sparse Decision Tree Optimization via Reference Ensembles. In *AAAI*.
[81] 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).
[82] 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*.
[83] Stephen H. Muggleton and Luc De Raedt. 1994. Inductive Logic Programming: Theory and Methods. *J. Log. Program.* 19/20 (1994), 629–679.
[84] Noel Novelli and Rosine Cicchetti. 2001. Fun: An efficient algorithm for mining functional and embedded dependencies. In *ICDT*. Springer, 189–203.
[85] 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.
[86] Thorsten Papenbrock and Felix Naumann. 2016. A Hybrid Approach to Functional Dependency Discovery. In *SIGMOD*.
[87] 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.
[88] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer.

2018. Deep contextualized word representations. In *NAACL*.
[89] Li Qian, Jinyang Gao, and H. V. Jagadish. 2015. Learning User Preferences by Adaptive Pairwise Comparison. *PVLDB* 8, 11 (2015), 1322–1333.
[90] J. Ross Quinlan. 1986. Induction of Decision Trees. *Mach. Learn.* 1, 1 (1986), 81–106.
[91] J. Ross Quinlan. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann.
[92] J. Ross Quinlan. 2004. Data Mining Tools See5 and C5.0.
[93] 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.
[94] Sanjay Rathee and Arti Kashyap. 2018. Adaptive-Miner: an efficient distributed association rule mining algorithm on Spark. *J. Big Data* 5 (2018), 6.
[95] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *EMNLP-IJCNLP*. 3980–3990.
[96] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *PVLDB* (2017).
[97] 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.
[98] Cynthia Rudin. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nat. Mach. Intell.* 1, 5 (2019), 206–215.
[99] 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.
[100] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR* abs/1910.01108 (2019).
[101] Hemant Saxena, Lukasz Golab, and Ihab F Ilyas. 2019. Distributed discovery of functional dependencies. In *ICDE*. IEEE, 1590–1593.
[102] Hemant Saxena, Lukasz Golab, and Ihab F Ilyas. 2019. Distributed implementations of dependency discovery algorithms. *PVLDB* 12, 11 (2019), 1624–1636.
[103] Philipp Schirmer, Thorsten Papenbrock, Ioannis Koumarelas, and Felix Naumann. 2020. Efficient Discovery of Matching Dependencies. *RODS* 45, 3 (2020), 1–33.
[104] 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*.
[105] Shaoyun Shi, Hanxiong Chen, Weizhi Ma, Jiaxin Mao, Min Zhang, and Yongfeng Zhang. 2020. Neural Logic Reasoning. In *CIKM*. 1365–1374.
[106] 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.
[107] Shaoxu Song and Lei Chen. 2009. Discovering matching dependencies. In *CIKM*.
[108] Ramakrishnan Srikant and Rakesh Agrawal. 1996. Mining Sequential Patterns: Generalizations and Performance Improvements. In *EDBT*. Springer, 3–17.
[109] 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.
[110] Saravanan Thirumuruganathan, Han Li, Nan Tang, Mourad Ouzzani, Yash Govind, Derek Paulsen, Glenn Fung, and AnHai Doan. 2021. Deep learning for blocking in entity matching: a design space exploration. *PVLDB* 14, 11 (2021), 2459–2472.
[111] Jeffrey D. Ullman. 2000. A Survey of Association-Rule Mining. In *International Conference on Discovery Science*. Springer, 1–14.
[112] Leslie G. Valiant. 1990. A Bridging Model for Parallel Computation. *Commun. ACM* 33, 8 (1990), 103–111.
[113] Flavian Vasile, Adrian Silvescu, Dae-Ki Kang, and Vasant G. Honavar. 2006. TRIPPER: Rule Learning Using Taxonomies. In *PAKDD*.
[114] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2021. Attention is All you Need. In *NeurIPS*. 5998–6008.
[115] Tong Wang, Cynthia Rudin, Finale Doshi-Velez, Yimin Liu, Erica Klampfl, and Perry MacNeille. 2017. A Bayesian Framework for Learning Rule Sets for Interpretable Classification. *J. Mach. Learn. Res.* 18 (2017), 70:1–70:37.
[116] G. I. Webb and S. Zhang. 2005. k-Optimal Rule Discovery. *Data Mining and Knowledge Discovery* 10, 1 (2005), 39–79.
[117] Renzhi Wu, Sanya Chaba, Saurabh Sawlani, Xu Chu, and Saravanan Thirumuruganathan. 2020. ZeroER: Entity Resolution using Zero Labeled Examples. In *SIGMOD*. 1149–1164.
[118] 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*.

[119] 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.
[120] H Yao, H Hamilton, and C Butz. 2002. Fd_mine: discovering functional dependencies in a database using equivalences, Canada. In *IEEE ICDM*. 1–15.
[121] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. 2017. Deep Sets. In *NeurIPS*. 3391–3401.
[122] Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. 1997. New Algorithms for Fast Discovery of Association Rules. In *KDD*. AAAI Press, 283–286.
[123] 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.
[124] Yunjia Zhang, Zhihan Guo, and Theodoros Rekatsinas. 2020. A Statistical Perspective on Discovering Functional Dependencies in Noisy Data. In *SIGMOD*. 861–876.
[125] 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.

Received July 2022; revised October 2022; accepted November 2022