Enriching Relations with Additional Attributes for ER

Mengyi Yan
Beihang University, China
yanmy@act.buaa.edu.cn

Wenfei Fan
Shenzhen Institute of Computing Sciences, China
University of Edinburgh
United Kingdom
Beihang University, China
wenfei@inf.ed.ac.uk

Yaoshu Wang
Shenzhen Institute of Computing Sciences, China
yaoshuw@sics.ac.cn

Min Xie[*]
Shenzhen Institute of Computing Sciences, China
xiemin@sics.ac.cn

ABSTRACT

This paper studies a new problem of relation enrichment. Given a relation D of schema R and a knowledge graph G with overlapping information, it is to identify a small number of relevant features from G, and extend schema R with the additional attributes, to maximally improve the accuracy of resolving entities represented by the tuples of D. We formulate the enrichment problem and show its intractability. Nonetheless, we propose a method to extract features from G that are diverse from the existing attributes of R, minimize null values, and moreover, reduce false positives and false negatives of entity resolution (ER) models. The method links tuples and vertices that refer to the same entity, learns a robust policy to extract attributes via reinforcement learning, and jointly trains the policy and ER models. Moreover, we develop algorithms for (incrementally) enriching D. Using real-life data, we experimentally verify that relation enrichment improves the accuracy of ER above 15.4% (percentage points) by adding 5 attributes, up to 33%.

PVLDB Reference Format:
Mengyi Yan, Wenfei Fan, Yaoshu Wang, and Min Xie. Enriching Relations with Additional Attributes for ER. PVLDB, 17(11): 3109 - 3123, 2024.
doi:10.14778/3681954.3681987

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/SICS-Fundamental-Research-Center/Enrichment.

1 INTRODUCTION

When we talk about incomplete information, we typically refer to (null) values and tuples missing from a relation D of schema R. However, for an application at hand, schema R may be incomplete; it "may not have all attributes required for analysis" [131]. As a consequence, the tuples in D do not have enough information for the application, *e.g.,* causal inference [131], aggregate SQL queries [84, 130] and movie popularity classification [33]. As exemplified in [131], a data analyst in the WHO organization aimed to estimate the effect of a mask policy on the coronavirus mortality rate, but found critical attributes (*e.g.,* weather that affects people's willingness to wear masks) not included in the data [131].

[*]Corresponding author

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.

Proceedings of the VLDB Endowment, Vol. 17, No. 11 ISSN 2150-8097.
doi:10.14778/3681954.3681987

Table 1: An Example of Person **Table**

tid	name	gender	email	address	city	age	spouse_name	eid
t_1	James Davis	M	james@example.com	18 Elmwood Rd	Chicago	45	Ava Davis	e_1
t_2	John Wilson	M	john@example.com	18 Maple Avenue	Houston	null	Ava Wilson	e_2
t_3	Ava Davis	F	ava@example.com	12 Pine Lane	Boston	36	John Wilson	e_3
t_4	Ava Wilson	F	a.Wat@example.com	18 Maple Avenue	Houston	36	John Wilson	e_3
t_5	Ava Davis	F	ad123@example.com	18 Elmwood Rd	Chicago	42	James Davis	e_4

Attributes are missing from schema R for several reasons [91, 114]. (1) People may lack full knowledge of desired functionality in large-scale applications. (2) The application world constantly evolves, necessitating enhancements. (3) The scale of tasks often requires incremental design and commissioning. Regardless of the reasons, relation D often misses features needed for our application.

In this paper, we focus on entity resolution (ER) as our target application, which has been commonly used in, *e.g.,* e-commerce and financial institutions. As an example, the insurance industry employs ER as a routine operation in detecting identity fraud, which was responsible for over \$17 billion stolen from U.S. consumers in 2017 [1]. As reported in [3], fraudsters forge fake identities. Such a fraud is hard to detect, since it can use information stolen from a real person, and a mix of real and synthetic data [6]. There are usually no unique identities as references, since collecting them is "time-consuming, adds friction to the customer journey, and is also an easy check to bypass" [12]. Similar problems also exist in card fraud (*e.g.,* duplicate applications for gift/credit cards) or money laundering (*e.g.,* move money from one account into another) [5, 111].

To detect fraud, the insurance industry has been advocating "to incorporate external information into the identity verification process" [6]. This has been practiced by, *e.g.,* SEON [13] and SIFT [14].

Example 1: Consider Table 1 with 5 tuples t_1-t_5 for 4 persons e_1-e_4, who are applying for promotional gift cards, where each person is limited to one card (*e.g.,* eGift of Starbucks [16]). The tuples have a schema with attributes name, gender, email, address and city; the store allows one person to have, *e.g.,* multiple different emails.

A not-so-sophisticated ER method $\mathcal{A}_{\mathsf{ER}}$ may predict that t_3 and t_5 are the same person (a false positive, FN), since they have exactly the same name. It may also miss the true match of t_3 and t_4 (a false negative, FN), due to different names, emails and addresses, although Ava Wilson was named Ava Davis before her marriage, and she issued a duplicate application using her old identity.

Fortunately, additional attributes can help us reduce FPs/FNs of ER. (a) As remarked in [131], age or ethnicity are often missing in person datasets. If additional attribute age is available, it provides a strong evidence for $\mathcal{A}_{\mathsf{ER}}$ to tell that t_3 and t_5 are mismatched, since they have different ages. (b) If we further enrich the schema with attribute spouse_name, (*e.g.,* by social networks [12]), $\mathcal{A}_{\mathsf{ER}}$ can identify t_3 and t_4, since they are married to the same person. □

Missing attributes are as damaging as missing data, but it has not received much attention. While there has been a host of work on missing data imputation [25, 26, 29, 40, 49, 50, 54, 55, 66, 77, 100, 105, 116, 128, 129], no prior work has systematically studied how to enrich incomplete schema in order to improve ER accuracy. They either do not target downstream applications or is not developed for ER and thus, miss distinguishing attributes for ER (see Section7).

To enrich dataset D of schema R for ER, we can extract information from external sources *e.g.,* text [63, 64], information space [43], XML [126], data warehouse [22] and Web [59]. In fact, knowledge enrichment has been practiced in medicine [112], network [133], e-commerce [15], recommendation [113] and text generation [132]. Among them, knowledge graphs (KGs) are particularly promising for ER. KGs link related entities and disambiguate entities with similar names [17]; moreover, their attributes are already reconciled and typically highly informative [130, 131], making it feasible to improve the ER accuracy. In particular, financial institutions have been using KGs to detect various types of fraud [5]; indeed, "using a KG for ER, the company can easily detect clusters of fake claims" [3].

In light of this, we study relation enrichment for ER by referencing a KG G. We consider *reliable* G, *i.e.,* a KG that is relatively clean and complete. Several popular KGs are in place, *e.g.,* general-purpose Wikidata [9] and domain-specific DRKG [71]. The issue is highly nontrivial. It requires us to link tuples in D with vertices in G for enrichment. Worse still, a KG typically maintains all sorts of properties of entities and their links to provide a comprehensive picture. If we enrich schema R with all such properties, it may hamper the accuracy of ER. As evidenced by [34], only *relevant* attributes contribute positively to identifying true positives, and "attributes containing null values may affect negatively the ER result". This motivates us to enrich schema with bounded relevant features.

Moreover, real-life datasets and KGs constantly change, *e.g.,* Wikidata publishes hundreds of live updates every minute [8] and the IMDB we used is refreshed daily [11]. In particular, financial institutions often require real-time detection of, *e.g.,* fraud in the online payment of credit cards [18]. It is too costly to conduct relation enrichment starting from scratch in response to the updates. These motivate the need for incremental enrichment.

To make the idea work, several questions have to be answered. What distinguishing features should we add to R to best improve the ER accuracy? For a tuple t in D, where can we find the additional attributes from KG G to complement t? How can we incrementally maintain the enriched D in response to updates to D and G?

Contributions & Organization. This paper studies relation enrichment for improving the accuracy of ER models. Consider a relation D of schema R and assume a reliable knowledge graph G.

(1) An enrichment scheme (Section 3). We formulate *the problem of relation enrichment for ER.* Given a black-box ER model $\mathcal{A}_{\mathsf{ER}}$ and a parameter m, it is to extract at most m features from knowledge graph G and extend schema R with the features as additional attributes, in order to maximize the accuracy of $\mathcal{A}_{\mathsf{ER}}$. We separate schema enrichment from data enrichment, and show that the former is NP-complete and the latter is in PTIME. This said, we propose a scheme ENRICH for enriching both schema R and relation D.

(2) Schema enrichment (Section 4). We propose a method for enriching schema R for ER under ENRICH. We develop a method for heterogeneous entity resolution (HER) to identify top-ranked matches (tuples and vertices) across relation D and KG G. We learn a policy via reinforcement learning to extract at most m features that extends R to schema R_G. Each feature is fetched by a path in G. We pick features that are as diverse from the existing attributes of R as possible, yield as few null values as possible, and maximumly improve the accuracy of ER. To make the policy robust to different data distributions, we jointly train the policy and the model $\mathcal{A}_{\mathsf{ER}}$.

(3) Data enrichment (Section 5). Under ENRICH, we develop algorithms for enriching the relation D to an instance D_G of schema R_G. We support both a batch mode and an incremental mode. In the batch mode, we extend each tuple t of D by identifying vertices v in G that refer to the same entity as t via HER, traversing paths from v to extract the additional features, and adding the features to t. In the incremental mode, we dynamically maintain D_G in response to updates to both relation D and graph G. To scale with large G and D, we parallelize the algorithms and show their parallel scalability, *i.e.,* they guarantee to reduce runtime when more resources are used [81]. We defer the parallelization to [10] for the lack of space.

(4) Experimental study (Section 6). Using real-life data and benchmarks, we empirically find the following. (a) Relation enrichment improves the accuracy of ER models by 15.4% on average, up to 33%, by adding 5 attributes. (b) It is on average 5.2% (resp. 14.6%) more accurate than ML models for feature augmentation (resp. feature selection), up to 18.2% (resp. 32.6%). (c) Batch enrichment is 5.94X faster than the baselines on IMDB on average. (4) The incremental method beats the batch one when updates to D and G are up to 20%, and is 6.28X faster when updates $|\Delta G| = 5\%|G|$.

We discuss related work in Section 7 and future work in Section 8.

2 PRELIMINARIES

In this section, we review basic notations, ER and HER.

Relations. Consider a relation schema $R = (\mathsf{id}, A_1, \ldots, A_n)$, where A_i is an attribute ($i \in [1, n]$), and id is an entity id as introduced by Codd [39], such that each tuple of R represents an entity of type τ with identity id. A relation D of R is a set of tuples of schema R.

Knowledge graphs. Following [68], we represent a knowledge graph as $G = (V, E, L)$. Here (a) V is a finite set of vertices representing entities, (b) $E \subseteq V \times V$ consists of edges representing relationships between entities; and (c) for each vertex $v \in V$, $L(v)$ is its feature or value, and for each edge $e \in E$, $L(e)$ is its label. Between a pair (v, v') in V, there are possibly multiple edges carrying distinct labels.

A *path* ρ *from a vertex* v_0 in graph G is $\rho = (v_0, v_1, \ldots, v_l)$ such that (v_{i-1}, v_i) is an edge in E for $i \in [1, l]$. The *length of* ρ is the number l of edges on ρ. A path is *simple* if each vertex appears on ρ at most once. We consider simple paths, simply referred to as paths.

Entity resolution. Given a relation D, entity resolution (ER) is to identify all pairs of tuples in D that refer to the same real-life entity. It returns a set of pairs (t_1, t_2) of tuples of D that are identified as *matches.* If t_1 does not match t_2, (t_1, t_2) is referred to as a *mismatch.*

A number of ER methods have been developed, based on ML [23, 45, 74, 86, 95, 102, 134], logic rules [24, 27, 32, 48, 61, 79, 127] and hybrid of the two [28, 41, 56]. We focus on ML-based ER models, which take quadratic-time (after the relevant ML models are trained).

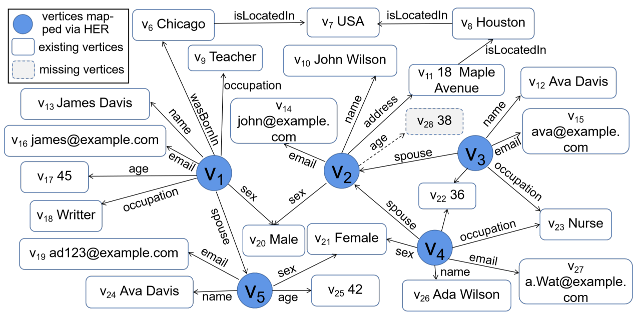


Figure 1: An example knowledge graph

Heterogeneous entity resolution (HER). HER is to identify entities across a relation and a graph. It is defined as a *mapping* f_{HER} that given a graph G and a set D of tuples of schema R, computes a set:

$$f_{\text{HER}}(D,G) = \{(t,v) \mid t \in D, v \in V \text{ in } G, t \Rightarrow v\}.$$

Here $t \Rightarrow v$ denotes that tuple t and vertex v make a *match*, *i.e.,* t and v refer to the same entity. For each $t \in D$, there can be multiple vertices v in G that match t. We refer to f_{HER} as the HER *mapping*.

Several methods are in place for HER, *e.g.,* rule-based JedAI [98], parametric simulation [51], and ML-based Silk [72], MAGNN [58] and EMBLOOKUP [19]. In particular, it is in $O(|D||G|)$ time to compute all HER matches across D and G by parametric simulation [51].

We assume that schema R has enough information for a well-designed HER mapping f_{HER} to map tuples of R to entities in G.

Example 2: Consider a KG G in Figure 1 for tuples in Table 1 (*e.g.,* derived from social networks [65]), where t_i matches v_i ($i \in [1,5]$).

HER differs from ER. (a) It identifies tuples and vertices across a relation and a graph, while ER matches tuples in a relation; (b) an attribute in t may map to a path in G, *e.g.,* city of t_2 vs. $\rho = (v_2, v_{11}, v_8)$ in G; (c) not every attribute in t can find a matching path in G, and vice versa, *e.g.,* v_4 has occupation "Nurse", which finds no corresponding attribute in Table 1; and (d) t and v often have different descriptions for the same property, *e.g.,* gender of t_2 vs. sex of v_2. □

3 A SCHEME FOR RELATION ENRICHMENT

In this section, we first formulate the enrichment problem and incremental enrichment problem for ER (Section 3.1). We then settle the complexity of the enrichment problems (Section 3.2). After this, we propose enrichment scheme ENRICH (Section 3.3).

3.1 Relation Enrichment Problem

Given a relation schema $R = (\bar{A})$, where $\bar{A}$ is a set $(\text{id}, A_1, \ldots, A_n)$ of attributes, consider a relation D of R, a KG G, and an ER model $\mathcal{A}_{\text{ER}}$.

Accuracy. We first present how to measure the accuracy improvement on ER models, in terms of Precision, Recall and F_1.

Following Codd [39], consider tuples for representing a (countably infinite) set $\mathcal{E}$ of real-world entities. For tuples t in instance D of schema $R = (\bar{A})$, there exists a mapping f from each tuple ID in D to $\mathcal{E}$ such that $f(t.\text{id}) = e$, *i.e.,* for each tuple t in D, $f(t.\text{id})$ is the entity represented by t (such mapping is usually implicitly assumed in ER). Then the accuracy of $\mathcal{A}_{\text{ER}}$ on D is traditionally measured in terms of $\mathsf{F}_1 = \frac{2\cdot\text{Precision}\cdot\text{Recall}}{\text{Precision}+\text{Recall}}$. Here Precision is the ratio of *pairs of distinct tuples* that are correctly identified to all identified tuple pairs, *i.e.,* Precision $= \frac{|\{(t,s)\ \mid\ t,s\in D,\ \mathcal{A}_{\text{ER}}(t,s)=\text{true},\ f(t.\text{id})=f(s.\text{id})\}|}{|\{(t,s)\ \mid\ t,s\in D,\ \mathcal{A}_{\text{ER}}(t,s)=\text{true}\}|}$ for distinct t and s, and Recall is the ratio of correctly identified tuple pairs to all tuple pairs that refer to the same real-world entity,

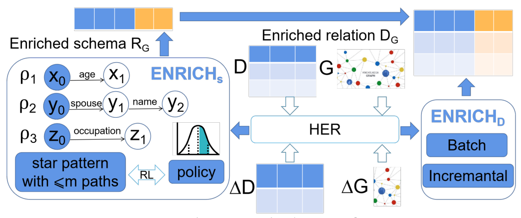


Figure 2: The workflow of ENRICH

i.e., Recall $= \frac{|\{(t,s)\ \mid\ t,s\in D,\ \mathcal{A}_{\text{ER}}(t,s)=\text{true},\ f(t.\text{id})=f(s.\text{id})\}|}{|\{(t,s)\ \mid\ t,s\in D,\ f(t.\text{id})=f(s.\text{id})\}|}$.

Example 3: Consider Table 1 of schema $R = (\bar{A})$ = (name, gender, email, address, city), where D has 5 tuples, $t_1[\bar{A}]$-$t_5[\bar{A}]$, and the mapping f is shown in the table. Here assume that $\mathcal{A}_{\text{ER}}$ makes an FP prediction and an FN prediction as stated in Example 1. The precision of $\mathcal{A}_{\text{ER}}$ on D is Precision $= \frac{0}{1}$ since the only distinct pair predicted true by $\mathcal{A}_{\text{ER}}$ is (t_3, t_5) (which is a FP). Similarly, Recall $= \frac{0}{1}$ since the only true match (t_3, t_4) is not identified (due to the FN). □

As shown above, $\mathcal{A}_{\text{ER}}$ on D is not accurate, for the lack of attributes. To improve it, we aim to enrich schema $R = (\bar{A})$ to $R_G = (\bar{A}, \bar{B})$, where $\bar{A}$ copies the attributes of R, $\bar{B}$ is a set of at most m attributes extracted from graph G, and m is the "budget" for extending schema R. Intuitively, we want to extend D and create an instance D_G of schema R_G, such that for each t in D, we have exactly one *enriched tuple* $t_G \in D_G$, where $t_G.\text{id} = t.\text{id}$, $t_G[\bar{A}] = t[\bar{A}]$ and $t_G[\bar{B}]$ is the partial tuple extracted from G. We refer to R_G as the *enriched schema* of R with G, and to D_G as the *enriched relation* of D with G.

Example 4: Assume that $R_G = (\bar{A}, \bar{B})$ where $\bar{B}$ = (spouse_name). After enriching R to R_G with an additional attribute spouse_name, the FN (*i.e.,* (t_3, t_4)) is reduced, as stated in Example 1, improving Precision of $\mathcal{A}_{\text{ER}}$ on D_G to $\frac{1}{2}$, since $\mathcal{A}_{\text{ER}}$ predicts true for both (t_3, t_5) (which is the FP) and (t_3, t_4), where only the latter one is correctly identified. Similarly, Recall is also improved to $\frac{1}{1}$ since the only true match (t_3, t_4) in Table 1 is correctly identified. □

We use the difference between the F_1 of $\mathcal{A}_{\text{ER}}$ on D_G and on D, denoted by $\Delta\mathsf{F}_1$, as *improvement of* $\mathcal{A}_{\text{ER}}$ *on* D *via* D_G. The difference of Precision/Recall can also be used, depending on application needs.

Problems. We now state the *enrichment problem* for ER model $\mathcal{A}_{\text{ER}}$.

- *Input*: $R = (\bar{A})$, D and G as above, and a positive integer m.
- *Output*: (a) An enriched schema $R_G = (\bar{A}, \bar{B})$ of R with G such that R is extended with at most m attributes $\bar{B}$ extracted from G; and (b) an enriched relation D_G of D with G.
- *Objective*: To maximize the improvement of $\mathcal{A}_{\text{ER}}$ on D via D_G.

Here ER is conducted by the same $\mathcal{A}_{\text{ER}}$ on both D and D_G.

The enrichment problem can be sub-divided into two problems: (1) *schema enrichment*, to deduce enriched schema R_G, and (2) *data enrichment*, to compute enriched relation D_G after R_G is in place.

Incremental enrichment problem. Real-life data is constantly changed by small updates. Consider updates to D and G. Updates to relation D consist of deleted/inserted tuples, denoted by ΔD; note that modifications to a tuple t can be regarded as deleting t followed by inserting a tuple with the changed values. Graph updates, denoted by ΔG, consist of edge insertions/deletions. Note that vertex updates are a dual [80] and can be handled similarly; and the change to an edge label can be seen as the deletion of an existing edge, followed by the insertion of a new one with the updated label. We use $G \oplus \Delta G$ to denote graph G updated by ΔG; similarly for $D \oplus \Delta D$.

When D and G are updated by ΔG and ΔD, respectively, the enriched relation D_G has also to be updated. In practice, ΔG and ΔD are often small. Hence we want to compute changes ΔD_G such that $D_G \oplus \Delta D_G$ is precisely the enriched relation of $D \oplus \Delta D$ with $G \oplus \Delta G$. The rational is that when ΔG and ΔD are small, so often is ΔD_G; hence it is more efficient to compute ΔD_G than to recompute the enriched relation of $D \oplus \Delta D$ with $G \oplus \Delta G$ starting from scratch. On the other hand, when ΔG and ΔD are small, schema R_G often remains unchanged and does not have to be recomputed. Hence we focus on computing ΔD_G in response to ΔD and ΔG after R_G is in place.

This motivates us to study the *incremental enrichment problem*.

- *Input*: R, D and G as above, and updates ΔD to D and ΔG to G.
- *Output*: Updates ΔD_G such that $D_G \oplus \Delta D_G$ is equal to the enriched relation of relation $D \oplus \Delta D$ with graph $G \oplus \Delta G$.
- *Objective*: Maximumly improve $\mathcal{A}_{\text{ER}}$ on $D \oplus \Delta D$ via $D_G \oplus \Delta D_G$.

Example 5: Continuing with Example 4, if $m = 2$, we can further extend schema R of Table 1 to R_G with attribute age from G of Figure 1, and improve Precision and Recall to 1 (the computation is similar). However, since age of entity e_2 is missing in G, the enriched tuple of t_2 has value null on age. When G is updated by adding a new edge $e = (v_2, v_{28})$ with $L(e)$ = age and $L(v_{28}) = 38$ (shown dashed in Figure 1), incremental enrichment dynamically updates D_G of R_G, by setting the age-value of the enriched tuple of t_2 to 38. □

3.2 Complexity of the Enrichment Problems

We next settle the complexity. We show that schema enrichment is NP-complete; in contrast, data and incremental enrichment are tractable, *i.e.,* in polynomial time (PTIME), after R_G is in place.

Theorem 1: *(1) The enrichment problem and schema enrichment problem are* NP*-complete. (2) The data enrichment problem and incremental enrichment problems are in* PTIME. □

Proof sketch: Below we show statement (1). We develop PTIME algorithms in Section 5 as a constructive proof for statement (2).

The decision problem of schema enrichment is to decide, given $R = (\bar{A})$, D, G, m, and a predefined threshold σ, whether there exists a set $\bar{B}$ of m attributes such that instance D_G of $R_G = (\bar{A}, \bar{B})$ has accuracy improvement of $\mathcal{A}_{\text{ER}}$ above the threshold σ.

(1) The upper bound is verified by first guessing m attributes for $\bar{B}$, and then computing D_G and checking whether the accuracy improvement is above σ; the computing and checking steps are in PTIME (to be verified in Section 5); hence the algorithm is in NP. Thus the enrichment problem is in NP; so is schema enrichment.

(2) We show that schema enrichment is NP-hard for ML-based ER and HER methods by reduction from X3C, which is NP-complete (cf. [60]). X3C is to decide, given a set H of elements with $|H| = 3q$ and a collection C of 3-element subsets of H, whether there exists an exact cover of H, *i.e.,* a sub-collection $C' \subseteq C$ such that each element in H is in exactly one set $S_i \in C'$. We show the NP-hardness also holds for rule-based ER and HER methods (see [10] for details). □

3.3 A Scheme for Enrichment

Despite the intractability, we propose a scheme for relation enrichment for (black box) ER model $\mathcal{A}_{\text{ER}}$, denoted as ENRICH.

As shown in Figure 2, ENRICH has two modules ENRICH_S and ENRICH_D for schema and data enrichment, respectively.

Schema enrichment. Given $R = (\bar{A})$, a reliable KG G, a training set S of tuples of schema R and a positive number m, ENRICH_S is to compute enriched schema $R_G = (\bar{A}, \bar{B})$ with at most m additional attributes. For each attribute $B \in \bar{B}$, it also returns a path ρ_B such that for each tuple t of R, the value of $t[B]$ can be fetched via path ρ_B in G from some vertices v that match t by HER. ENRICH_S is conducted *once offline*, *i.e.,* we re-use R_G for each input relation D of R.

Data enrichment. After schema $R_G = (\bar{A}, \bar{B})$ is computed from ENRICH_S, ENRICH_D populates and dynamically maintains relation D_G of R_G online. It supports the following two modes.

(1) Batch mode: Given schema R_G, a relation D of schema R and a KG G, ENRICH_D generates relation D_G of R_G. For each tuple t in D, we find its HER matches, *i.e.,* a set of vertices v in G, and create an enriched tuple of R_G for t. As t and v refer to the same entity, we complement t with $B \in \bar{B}$ features of v if the features are available in G.

(2) Incremental mode: ENRICH_D incrementally maintains D_G in response to updates ΔD and ΔG online. Updates may change not only paths ρ_B corresponding to attributes extracted, but also vertices in graphs that match tuples via HER. ENRICH_D dynamically computes changes ΔD_G to D_G, rather than starting from scratch.

4 SCHEMA ENRICHMENT

In this section, we learn an effective policy and develop an algorithm SchemaEnr for ENRICH_S. Consider a black box ER model $\mathcal{A}_{\text{ER}}$, differentiable or non-differential. Given schema $R = (\bar{A})$, a reliable KG G, a training set S of tuples of schema R and two numbers m and k, we compute an enriched schema $R_G = (\bar{A}, \bar{B})$ of R with G to maximumly improve the accuracy of $\mathcal{A}_{\text{ER}}$. Here $\bar{B}$ consists of at most m distinct attributes, along with a path pattern ρ_B of length at most k for each $B \in \bar{B}$. While a larger k may extract more features, it often leads to more null values and weaker semantic associations.

With a slight abuse of notations, we use the following notions.

- A *path pattern* has the form $\rho = (x_0, L_1, x_1, \ldots, x_{l-1}, L_l, x_l)$, where (1) each x_i ($i \in [1, l]$) is a distinct variable, and x_0 is referred to as the *center* of ρ, and (2) each (x_{i-1}, x_i) is an edge pattern with label L_i. As will be seen shortly, we use path patterns to locate features of the entity denoted by x_0.
- A *match* of path pattern ρ in G, denoted by $h(\rho)$, is a mapping h from ρ to G such that (1) for each variable x_i, $h(x_i)$ is a vertex in G, where $h(x_0)$ is the *pivot of the match*, and (2) for each edge pattern (x_{i-1}, x_i), $(h(x_{i-1}), h(x_i))$ is an edge in G with the same label L_i. Intuitively, $h(\rho)$ is a specific path from vertex $h(x_0)$ in G, and fetches the value of a selected feature (Section 5.1).

Naive algorithms. To build schema R_G, one may want to use a greedy strategy that iteratively picks an attribute to maximize the *mutual information*, or to train an ML model that retrieves m "relevant" ρ_B in G. These, however, do not work well, for three reasons: (a) The holistic effect of multiple attributes cannot easily be captured by mutual information. (b) There are an exponential number of paths in G and thus, it is too costly to enumerate them all and find m path patterns that maximize the accuracy. (c) It is hard to define an explicit loss for training of the *black-box* $\mathcal{A}_{\text{ER}}$.

Overview. In light of this, we adopt an approach based on policy-learning with a parameterized policy function π_θ (*i.e.,* θ is the set of parameters in π_θ). It consists of the following steps.

(1) HER mapping. Taking a set D of tuples of schema R and graph G as input, we pre-compute the set of HER matches in G.

(2) Policy learning and model fine-tuning. Given the HER matches obtained above, we interleave the policy learning and model training to *jointly* learn a policy π_θ, enrich R via *reinforcement learning (RL)*, and improve the accuracy of model $\mathcal{A}_{\mathsf{ER}}$ on the enriched data.

Below we present our HER mapping and policy function in Sections 4.1-4.2, respectively, and then give SchemaEnr (Section 4.3).

4.1 Heterogeneous Entity Resolution

We start with a method for the following HER problem.

- *Input:* $R = (\bar{A}), G$, a tuple t of schema R and a positive number K.
- *Output:* A set $\mathcal{V}_t$ of top-K vertices that match t (*i.e.,* refer to the same entity) and have the largest *correlation strengths* with t.

Intuitively, for a tuple t of schema R, there are possibly multiple vertices v in G that match t. Our HER method finds top-K matching vertices by two steps: (1) *blocking*, which retrieves candidate vertices in G for the given tuple t, and (2) *ranking*, which returns the top-K set $\mathcal{V}_t$ with the highest correlation strengths, via a ranking strategy.

Blocking. Given t, we initialize the set C_t of candidate matches as blocks, by taking all vertices v whose "similarity" to t are above a predefined threshold. We adopt the *Jaccard similarity.* More specifically, we serialize all values of t to a sequence and tokenize it into a set $\mathsf{Set}(t)$. For each vertex $v \in G$, we extract an induced subgraph G_v of G, including v and the neighbors of v. For each v_i in G_v, we serialize and tokenize its label $L(v_i)$. Denote the union of token sets of all vertices in G_v by $\mathsf{Set}(G_v)$. Then the Jaccard similarity is computed by $\mathsf{Jacc}(t, v) = \frac{|\mathsf{Set}(G_v) \cap \mathsf{Set}(t)|}{|\mathsf{Set}(G_v) \cup \mathsf{Set}(t)|}$. Intuitively, (t, v) makes a candidate match only if t and G_v share enough keywords.

After that, we compute HER matches within each C_t for each t.

Ranking. Jaccard similarity only considers syntactic similarity to form candidates. To find true HER matches, within each block, we identify vertices v that match t via *parametric simulation* (see more in [51]), which complements the blocking with *semantic checking*. Intuitively, it recursively checks the pairwise semantic closeness between the attributes of t and the descendants of v, by embedding ML in topological matching. For each $A \in \bar{A}$, we can find a path pattern ρ_A such that a match of ρ_A pivoted at v in G represents $t[A]$, *e.g.,* attribute city vs. path pattern $(x_0, \mathsf{address}, x_1, \mathsf{isLocatedIn}, \mathsf{x}_2)$.

We rank the HER matches and pick top-K ones as follows. We expand G_v by DFS following each path pattern ρ_A, starting from v. We adopt SentBert [104], a bert-based model, to transform each vertex v_i in G_v ($i \in [0, l]$) into an embedding $\mathbf{e}_{v_i}$. Similarly, we serialize t and use SentBert to transform it to an embedding $\mathbf{e}_t$. We measure the semantic similarity between t and its most relevant vertex in G_v, via $\mathsf{sem}(t, v) = \mathsf{max}_{v_i \in G_v} \mathsf{cos}(\mathbf{e}_t, \mathbf{e}_{v_i})$, where cos is the cosine similarity. We adopt the contrastive learning strategy [122] to pre-train SentBert by self-annotated training data.

Given K, we rank all matching vertices v and return the set $\mathcal{V}_t$ of top-K HER matches with the largest semantic correlation strengths.

4.2 Policy Function

We next present the objective and training of our policy function.

Representing $\bar{B}$. We represent $\bar{B}$ as a set Q of path patterns, *i.e.,* $Q = \{\rho_B \mid B \in \bar{B}\}$. Each $B \in \bar{B}$ is specified by a pattern ρ_B and $B = L_1 \dots L_l$, *i.e.,* its attribute name is the concatenation of edge labels of ρ_B.

Based on the path pattern $\rho_B (B \in \bar{B})$, for each tuple t of schema R, we can compute an enriched tuple t_G for t, by instantiating each B-attribute of t_G following the path matches of ρ_B pivoted at some vertices in the set $\mathcal{V}_t$ of top-ranked HER matches of t (see below).

Objective. Below are criteria for ρ_B ($B \in \bar{B}$). Consider a validation set T (resp. an enriched T_G of T) of schema R (resp. $R_G = (\bar{A}, \bar{B})$).

(1) Diversity. We adopt mutual information $\mathsf{MI}(x, y)$ [31] to measure the correlation between attributes x and y. We define the diversity of R_G on T_G as $\mathsf{div}(T_G) = -\frac{1}{|R_G|(|R_G|-1)} \sum_{x,y \in \bar{A} \cup \bar{B} \& x \neq y} \mathsf{MI}(x, y)$. Intuitively, we want to enrich $R = (\bar{A})$ with new attributes $\bar{B}$ that are as diverse as possible from each other and from the existing $\bar{A}$.

(2) Completeness. We count and normalize the number of null values in $\bar{B}$ on the validation relation T_G as the completeness, *i.e.,* $\mathsf{comp}(T_G) = -\frac{\#\{\text{null values}\}}{\#\{\text{all values}\}}$. Fewer null values are more desirable.

(3) Distinguishability. The enriched $\bar{B}$ should be distinguishing, improving the accuracy of $\mathcal{A}_{\mathsf{ER}}$ on T_G, denoted by $\mathsf{F}_1(T_G, \mathcal{A}_{\mathsf{ER}})$.

Taken together, the objective value we want to maximize is:
$\mathsf{obj}(T_G, \mathcal{A}_{\mathsf{ER}}) = w_{\mathsf{div}} \mathsf{div}(T_G) + w_{\mathsf{comp}} \mathsf{comp}(T_G) + w_{\mathsf{F}_1} \mathsf{F}_1(T_G, \mathcal{A}_{\mathsf{ER}})$
where w_{div}, w_{comp} and w_{F_1} are weights of the criteria, respectively.

Then the schema enrichment problem is equivalent to finding Q that maximizes $\mathsf{obj}(T_G, \mathcal{A}_{\mathsf{ER}})$. We approach it via *policy learning.*

Policy function. We iteratively construct the set Q of path patterns by building the patterns one by one, adding one edge at a time, via a parameterized policy π_θ, until all m path patterns are in place.

Given a (partially constructed) set Q, we can create an enriched tuple t_G for each t in D for computing the objective value mainly in the following three steps (see Section 5.1). (a) For each HER match v of t in $\mathcal{V}_t$, we instantiate the center x_0 of each ρ_B in Q by v. (b) Starting from v, we follow the edge labels in ρ_B to get a candidate value of $t_G[B]$. (c) Given all such candidate values, we employ a ranking model to assign the most promising value to $t_G[B]$.

Assume that we have $i-1$ paths $\rho_{B_1}, \dots \rho_{B_{i-1}}$ ($i \in [1, m]$), and the i-th path ρ_{B_i} is partially constructed with j edges ($j \in [1, k-1]$). Denote by $Q_{i,j}$ the resulting partial set, and by $T_{Q_{i,j}}$ the enriched relation of the validation set T under partial schema $(\bar{A}, B_1, \dots, B_i)$. We use the partial $Q_{i,j}$ as state $s_{i,j}$ and the next edge e to be added as action $a_{i,j}$. After taking action $a_{i,j}$, state $s_{i,j}$ is transmitted to a new state $s_{i,j+1} = Q_{i,j+1}$, which extends path pattern ρ_{B_i} with a new edge e. We add a special action [SEP] to terminate the expansion of ρ_{B_i}, and stop it if its length is k. In each step, we compute the improvement on the objective value as the reward $r_{i,j}$, *i.e.,*

$$r_{i,j} = \mathsf{obj}(T_{Q_{i,j+1}}, \mathcal{A}_{\mathsf{ER}}) - \mathsf{obj}(T_{Q_{i,j}}, \mathcal{A}_{\mathsf{ER}}).$$

Then we use a *policy function* π_θ with parameter θ to map each state $s_{i,j}$ to a vector $\mathbf{a}_{i,j}$ of action probabilities, *i.e.,* $\pi_\theta = p(a_{i,j} \mid s_{i,j}, \theta)$. We adopt a CNN neural network for π_θ and define

$$\mathbf{a}_{i,j} = \mathsf{softmax}(\mathsf{FC}(\mathsf{CNN}(\mathsf{transform}(s_{i,j})))),$$

where $\mathsf{transform}(s_{i,j})$ [70] computes a binary vector of state $s_{i,j} = Q_{i,j}$, and FC is a fully-connected layer.

To find the optimal Q, we learn π_θ to maximize the expected reward $\mathbb{E}_{p(s_{i,j};\theta)}[r_{i,j}]$. Here the reward can be non-differentiable because it is computed based on $\mathcal{A}_{\mathsf{ER}}$ in the validation data, and the action space is large. Thus, we use Maskable PPO [70, 107], the invalid action masking for the Proximal Policy Optimization method that imposes no constraints on $\mathcal{A}_{\mathsf{ER}}$, to iteratively update the set θ of parameters of π_θ with the following loss function $\mathcal{J}_\theta$:

$$\mathcal{J}_\theta = \sum_{x=1}^{i} \sum_{y=1}^{s_{x,\cdot}} \mathbb{E}_{p(s_{x,y};\theta_{\text{old}})} [\frac{p(a_{x,y}|s_{x,y-1};\theta)}{p(a_{x,y}|s_{x,y-1};\theta_{\text{old}})} \times \hat{A}_{\theta_{\text{old}}}(s_{x,y-1}, a_{x,y})] - \beta \times \mathsf{KL}(\theta, \theta_{\text{old}}) \quad (1)$$

where θ_{old} is the set of parameters before updates, $\hat{A}_{\theta_{\text{old}}}$ is an estimated advantage function computed by rewards [107], and KL is Kullback-Leibler Divergence that measures the distribution discrepancy between π_θ and $\pi_{\theta_{\text{old}}}$, and is a regularization term.

Intuitively, by adopting such a policy learning approach, we give path patterns low probabilities if their rewards (feedback) are negative or small, so that they are not selected in the next iterations. The policy gradually learns which edges are promising to add and only *relevant* attributes are enriched. If all remaining attributes are bad, the policy may stop enrichment and stick to the current attributes. Hence $R_G = (\bar{A}, \bar{B})$ is as least as good as $R = (\bar{A})$.

Example 6: Consider the path patterns in Figure 2. Assume that we have constructed $\rho_1 = (x_0, \text{age}, x_1)$, and $\rho_2 = (y_0, \text{spouse}, y_1)$ is partially constructed. Then we continually add more edges with the maximum reward, following π_θ. Suppose that we add (y_1, y_2) (labeled name) to ρ_2, followed by the special action [SEP]. We then terminate the expansion of ρ_2 and continue to construct other paths. □

4.3 Algorithm for Schema Enrichment

Although the policy π_θ is able to construct path patterns without costly enumeration, it stills encounters some issues. (1) The distributions of the training and validation sets keep changing due to schema enrichment, and it is costly to frequently re-train $\mathcal{A}_{\mathsf{ER}}$. Worse still, (2) the efficiency of policy learning depends on the feedback from $\mathcal{A}_{\mathsf{ER}}$; this makes the policy learning process expensive.

In light of these, we propose SchemaEnr for schema enrichment. Its novelty includes a joint training strategy for π_θ and $\mathcal{A}_{\mathsf{ER}}$, making up the time for computing feedbacks from $\mathcal{A}_{\mathsf{ER}}$ in policy learning.

Algorithm. Given schema R, a training (resp. validation) set S (resp. T) of tuples of schema R, a graph G, an ER model $\mathcal{A}_{\mathsf{ER}}$, a maximum batch number I, parameters m, k and K for constraining the maximum additional attributes, the length of path patterns and the number of HER matches, respectively, we give SchemaEnr in Figure 3. It returns enriched $R_G = (\bar{A}, \bar{B})$ such that the objective value is maximized on the enriched validation data. Here S can be obtained from benchmarks or by manual labeling a few candidates (see Section 6).

After initializing π_θ (line 1), SchemaEnr pre-computes the top-K HER matches in G for each tuple in S or T, (lines 2-3). Following [92], we *jointly* optimize π_θ and $\mathcal{A}_{\mathsf{ER}}$ in *batches* (line 4-18) such that the policy function learns to find "good" path patterns and the ER model is fine-tuned to improve the accuracy simultaneously.

Joint training. In each batch, the training set, the validation set and the set of additional attributes for the current batch are denoted by S_{train}, T_{valid} and $\bar{B}^{\text{bat}}$, respectively; $\bar{B}^{\text{bat}}$ is empty initially (lines 5-6).

Policy π_θ is first fixed and the set $\bar{B}^{\text{bat}}$ is constructed iteratively to train $\mathcal{A}_{\mathsf{ER}}$ (lines 7-12). In the i-th iteration, a new path $\rho^{\text{bat}}_{B_i}$ is located based on the policy π_θ, via procedure PathPolicy (omitted). Intuitively, it continually adds a new edge with the maximum reward following π_θ until either [SEP] is added or $|\rho^{\text{bat}}_{B_i}| > k$. A new attribute B^{bat}_i is created accordingly by concatenating the edge labels of $\rho^{\text{bat}}_{B_i}$ (line 8). Note that even when one more attribute is added, the distribution of enriched data may change dramatically. To make $\mathcal{A}_{\mathsf{ER}}$ robust to diverse distributions, we accumulate the enriched training (resp. validation) data in a set $\mathcal{S}_{\text{train}}$ (resp. $\mathcal{T}_{\text{valid}}$), which are initially empty (line 5), during the iterative process (lines 9-11). Whenever we get a new attribute B^{bat}_i, we compute the enriched relations of S_{train} and T_{valid} (see Section 5.1) and add them to $\mathcal{S}_{\text{train}}$ and $\mathcal{T}_{\text{valid}}$, respectively. Finally, the entire $\mathcal{S}_{\text{train}}$ is adopted to upgrade $\mathcal{A}_{\mathsf{ER}}$ with the cross entropy loss (line 12).

Then we fix $\mathcal{A}_{\mathsf{ER}}$ and learn π_θ by iteratively sampling path patterns (line 13), via procedure SampleQ (see below), and update the parameter θ of π_θ based on the advantage function $\hat{A}_\theta$ and the loss $\mathcal{J}_\theta$ (Line 14 - 17, see below). Intuitively, at each state $s_{i,j}$, the next action is sampled from the action probabilities of π_θ.

Both π_θ and $\mathcal{A}_{\mathsf{ER}}$ are optimized iteratively until it reaches the maximum number I of batches. Finally, we obtain the final set $\bar{B}$, by calling procedure Inference (omitted), which performs actions with maximum rewards following π_θ (see [10]). With a small learning rate α and consistently convergent $\mathcal{A}_{\mathsf{ER}}$, π_θ will eventually converge, and at least to a local minima [99, 125], *e.g.,* in Section 6, SchemaEnr only needs approx. 5 iterations to converge on average.

Procedure SampleQ. Taking current policy π_θ as input, SampleQ samples a set of path patterns as Q following the action probabilities of π_θ. To enable effective sampling, we design a *mask* strategy. When selecting path patterns, we filter out those with small completeness, *e.g.,* less than 10%. Attributes with many null values are considered as low quality and π_θ need not to explore them.

Procedure Reward. Given the current state s, Reward computes its reward r_s (Section 4.2). Since $\mathcal{A}_{\mathsf{ER}}$ is not stable in the first few epochs, we design a *warm-up* strategy, which sets a small weight w_{F1} for F_1 and a large weight w_{div} (resp. w_{comp}) for div (resp. comp) so that π_θ is not affected by unstable $\mathcal{A}_{\mathsf{ER}}$. Then w_{F1} (resp. w_{div} and w_{comp}) gradually increases (resp. decrease) until they become 1. After computing the reward r_s, we compute the advantage function $\hat{A}_\theta$ based on r_s using θ from the previous iteration.

Example 7: Consider the tuples from Table 1 as the training set S of tuples, with $m = 2$ and $k = 2$. In the first iteration, $\mathcal{A}_{\mathsf{ER}}$ is first trained on S. Due to the lack of initial attributes, $\mathcal{A}_{\mathsf{ER}}$ does not work well. Then SchemaEnr executes SampleQ to sample a few path patterns, *e.g.,* $\rho_1 = (x_0, \text{age}, x_1)$, $\rho_2 = (y_0, \text{spouse}, y_1, \text{name}, y_2)$, $\rho_3 = (z_0, \text{occupation}, z_1)$ and $\rho_4 = (u_0, \text{wasBornIn}, u_1, \text{isLocatedIn}, u_2)$. Suppose $\{\rho_1, \rho_4\}$ is sampled. When ρ_1 is added into $\bar{B}$, the reward is 0.5. However, when ρ_4 is added, the reward drops to 0.4 since the values of the ρ_4-attribute of all tuples are null except t_1. Thus in the next iteration, π_θ gives higher (resp. lower) probability for ρ_1 (resp. ρ_4) to be sampled. To balance exploration and exploitation, π_θ also gives certain probabilities for unseen paths, *e.g.,* ρ_3. After several iterations, π_θ is learned to select good path patterns and $\mathcal{A}_{\mathsf{ER}}$ is fine-tuned to adapt to data with different schema. Finally $\{\rho_1, \rho_2\}$ is sampled and SchemaEnr finds the "optimal" $\bar{B}$. □

Complexity. SchemaEnr is in $O((|S| + |T|)|G|Imk)$ time, when it takes I batches to train π_θ and $\mathcal{A}_{\mathsf{ER}}$. The HER mapping takes

```
Input: A schema $R = (\bar{A})$, a training set $S$, a validation set $T$, a graph $G$,
    $\mathcal{A}_{\text{ER}}$, a batch number $I$ and parameters $m$, $k$ and $K$.
Output: An enriched schema $R_G = (\bar{A}, \bar{B})$.
1.  Initialize the policy function $\pi_\theta$; bat := 0;
2.  for each $t$ in $S$ or $T$ do
3.      $\mathcal{V}_t$ := the top-$K$ HER matches of $t$ in $G$;
4.  while bat $< I$ do
5.      $\mathcal{S}_{\text{train}}$ := getBatch$(S)$; $T_{\text{valid}}$ := getBatch$(T)$; $\mathcal{S}_{\text{train}} := \mathcal{T}_{\text{valid}} := \emptyset$;
6.      $\bar{B}^{\text{bat}} := \emptyset$; /* The enriched schema for the current batch */
        /*Joint training: Fix policy $\pi_\theta$ and train $\mathcal{A}_{\text{ER}}$ */
7.      for each $i \in [1, m]$ do
8.          $\rho_{B_i}^{\text{bat}}$ := PathPolicy$(\bar{A}, \bar{B}^{\text{bat}}, \pi_\theta)$; $\bar{B}^{\text{bat}} := \bar{B}^{\text{bat}} \cup \{B_i^{\text{bat}}\}$;
            /* Compute enriched relations based on HER matches in $\mathcal{V}_t$ */
9.          $\Delta_{\text{train}}$ := the enriched relation of $\mathcal{S}_{\text{train}}$ under schema $(\bar{A}, \bar{B}^{\text{bat}})$;
10.         $\Delta_{\text{valid}}$ := the enriched relation of $\mathcal{T}_{\text{valid}}$ under schema $(\bar{A}, \bar{B}^{\text{bat}})$;
11.         $\mathcal{S}_{\text{train}} := \mathcal{S}_{\text{train}} \cup \Delta_{\text{train}}$; $\mathcal{T}_{\text{valid}} := \mathcal{T}_{\text{valid}} \cup \Delta_{\text{valid}}$;
12.     Upgrade $\mathcal{A}_{\text{ER}}$ with gradient $\nabla_{\mathcal{A}_{\text{ER}}}$CrossEntropy$(\mathcal{S}_{\text{train}})$;
        /*Joint training: Fix $\mathcal{A}_{\text{ER}}$ and learn policy $\pi_\theta$ */
13.     $Q$ := SampleQ$(\pi_\theta)$ where $Q$ has $m$ paths $\rho_{B_1}, \ldots, \rho_{B_m}$;
14.     for each state $s_{i,j} = Q_{i,j}$ when generating $Q$ with $\pi_\theta$ do
15.         rw_sum := $\sum_{s=|Q_{i,j}|}^{|Q|} \gamma^{I-|Q_{i,j}|} \cdot r_s$, where $\gamma$ is the decay
                factor and $r_s$ is the reward at state $s$, i.e., $r_s$ = Reward$(s)$;
16.         Compute the advantage function $\hat{A}_\theta$ according to rw_sum;
17.         Update $\theta$ by optimizing $\mathcal{J}_\theta$ (Equation 1) with learning rate $\alpha$;
18.     bat := bat + 1;
19. $\bar{B}$ := Inference$(\bar{A}, \pi_\theta)$;
20. return $R_G = (\bar{A}, \bar{B})$;
```

Figure 3: Algorithm SchemaEnr

$O((|S| + |T|)|G|)$ time. In each epoch, it generates $\mathcal{S}_{\text{train}}$ and $\mathcal{T}_{\text{valid}}$ in $O((|S| + |T|)mk)$ time; moreover, π_θ takes $O(|T|mk)$ time to sample and learn, and $\mathcal{A}_{\text{ER}}$ typically takes $O(|S| + |T|)$ time to train and fine-tune. As will be seen in Section 6, our joint training strategy reduces the cost by making up the time for fine-tuning $\mathcal{A}_{\text{ER}}$, *e.g.*, it takes 2,213s to learn the policy on 3,162 tuples in 10 epoches.

5 POPULATING ENRICHED SCHEMA

Below we develop algorithms for populating and maintaining relations D_G of schema R_G after $R_G = (\bar{A}, \bar{B})$ is computed (along with the path pattern ρ_B for each B in $\bar{B}$). We develop a batch algorithm BEnrich (Section 5.1) and an incremental IncEnrich (Section 5.2), parallelized as PBEnrich and PIncEnrich, respectively [10].

5.1 Batch Enrichment

Algorithm BEnrich mainly consists of two steps: (1) HER *mapping*, which retrieves the set $\mathcal{V}_t$ of top-K HER matches for each t in D (presented in Section 4.1); and (2) *Populating*, which instantiates the $\bar{B}$-attribute values to get the enriched relation D_G (see below).

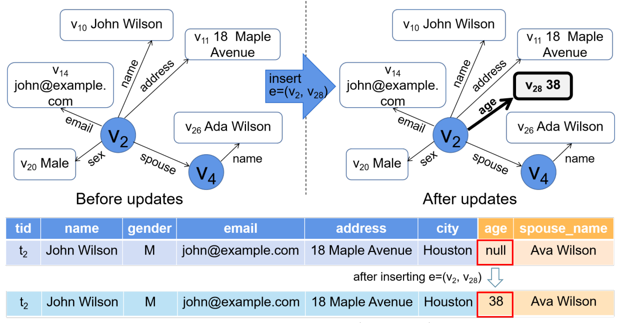
Populating. For each t, we create an enriched tuple t_G as follows.

(a) For each $A \in \bar{A}$, $t_G[A]$ copies the corresponding $t[A]$; and

(b) For each $B \in \bar{B}$, we compute a set $C_{t_G[B]}$ of candidate values for $t_G[B]$ and use a ranking model $\mathcal{M}_{\text{rank}}$ to assign the top-ranked one to $t_G[B]$ (see [10]); we set $t_G[B]$ = null if $C_{t_G[B]}$ is empty.

Generating candidate values. Initially, the set $C_{t_G[B]}$ is empty. For each HER match v of t in $\mathcal{V}_t$, we use the path matches h of pattern ρ_B pivoted at v to generate candidate values of $t_G[B]$, *i.e.*, for each path match h, we add the label of the last vertex of h to $C_{t_G[B]}$.

There is a trade-off between the length of paths and the number of null values. On the one hand, a longer ρ_B may lead to more combinations of edge labels and thus, more candidate attributes B. On the other hand, it is harder to find a path match of a longer ρ_B, and the B-attribute values of more tuples may set null, if we cannot find such path matches. To strike a balance, we use the parameter k to bound the length of paths, which will be tested in Section 6.



tid	name	gender	email	address	city	age	spouse_name
t_2	John Wilson	M	john@example.com	18 Maple Avenue	Houston	null	Ava Wilson

t_2	John Wilson	M	john@example.com	18 Maple Avenue	Houston	38	Ava Wilson

Figure 4: Incremental enrichment

Example 8: Given the path patterns in Figure 2, the HER mapping step links t_i in Table 1 to v_i in Figure 1 for $i = 1, \ldots, 5$. The populating step then traverses all path matches pivoted at v_i in G and fills in the values of the enriched attributes of t_i in D_G, *e.g.*, $h_1 : \{(v_5, v_{25}) \mapsto (x_0, x_1)\}$ is the only path match of ρ_1 pivoted at v_5 and thus, the age-value of the enriched tuple of t_5 is 42 by the ranking model. In contrast, ρ_3 finds no path match pivoted at v_5 and thus, the occupation-value of the enriched tuple of t_5 is null. $\square$

Complexity. Since we traverse paths to populate enriched schema, BEnrich takes $O(|D||G| + |D||C_{\max}|Km)$ time, where K is the maximum number of HER matches for each t in D and $|C_{\max}|$ is the maximum number of candidate values for a given attribute and a given HER match of t. Thus BEnrich is in PTIME; this constructively proves PTIME data enrichment and checking for Theorem 1.

5.2 Incremental Enrichment

We next develop the incremental algorithm IncEnrich.

Setting. We consider both graph updates ΔG and relation updates ΔD, where ΔD consists of deleted/inserted tuples and ΔG consists of edges. The goal is to compute ΔD_G such that $D_G \oplus \Delta D_G$ is equal to the enriched relation of relation $D \oplus \Delta D$ with graph $G \oplus \Delta G$.

We can divide ΔD_G into two parts: (a) the enriched relation of ΔD with $G \oplus \Delta G$, and (b) the updates of the enriched relation of D with $G \oplus \Delta G$. For part (a), it can be directly applying the batch algorithm to ΔD with $G \oplus \Delta G$. Below we mainly focus on part (b).

Recall that in the enriched schema $R_G = (\bar{A}, \bar{B})$, each attribute $A \in \bar{A}$ is also associated with a path pattern ρ_A. When G is updated, the path matches of ρ_A (and thus HER matches) may also change, a complication introduced by incremental enrichment.

Auxiliary structures. We maintain the following for incremental enrichment: (1) $\mathcal{V}_t$, the set of top-K HER matches for each t in D; (2) C_t, the set of all qualified vertices after blocking for each t in D, to allow efficient updates on the top-K ones, (3) Piv, an inverted index that maps each edge e in G to a list of pivots v_0 in G, such that there exists a path match h of pattern ρ_A (resp. ρ_B) pivoted at v_0, and e is an edge of path $h(\rho_A)$ (resp. $h(\rho_B)$); intuitively, Piv helps us identify pivots that can be affected by e. (4) Indices to get HER matched vertices (resp. tuples) for each t in D (resp. each v in G).

```
Input: An enriched relation $D_G$, a knowledge graph $G$, schema $R_G = (\bar{A}, \bar{B})$,
       graph updates $\Delta G$ and the auxiliary structures.
Output: The updates of the enriched relation of $D$ with $G \oplus \Delta G$.
1.  $P$ := GetAffectedPathMatches($\Delta G, G$, Piv);
2.  Group the path matches in $P$ by pivots;
3.  for each $h \in P_{v_0}$, where $P_{v_0}$ stores affected matches pivoted at $v_0$ do
4.      if $h$ is a path match of $\rho_B$ where $B \in \bar{B}$ do /* Case [C1] */
5.          for each $t$ whose top-$K$ HER matches include $v_0$ do
6.              Update the $B$-attribute value of the enriched tuple $t_G$ of $t$;
7.      if $h$ is a path match of $\rho_A$ where $A \in \bar{A}$ do /* Case [C2] */
8.          for each $t$ such that $v_0$ is in $\mathcal{V}_t$ or $C_t$ do
9.              Re-compute $\mathcal{V}_t$ and $C_t$;
10.             if $\mathcal{V}_t$ is updated do
11.                 Re-populating all $\bar{B}$-attribute values of $t_G$ based on $\mathcal{V}_t$;
12. return $\Delta D_G = \{t_G \in D_G \mid t_G$ is updated$\}$;
```

Figure 5: Algorithm IncEnrich

Incremental algorithm. We first incrementalize BEnrich with unit updates (*i.e.,* insertion/deletion of an edge). Then we show how to process a batch update ΔG (*i.e.,* a sequence of unit updates) to G.

Unit insertion. When an edge e is inserted into G, we create an new entry, denoted by Piv(e), and initialize it to be empty. Then we traverse the path matches h of ρ_A/ρ_B of $R_G = (\bar{A}, \bar{B})$ that pass through e, and add the pivot v_0 of h to Piv(e). We group these path matches by their pivots, and use P_{v_0} to denote the set of all new path matches pivoted at v_0 that are generated due to the insertion of e.

We process each path match h in P_{v_0} in the following two cases.

(1) **[C1] When h is a path match of ρ_B where $B \in \bar{B}$.** In this case, edge updates on $h(\rho_B)$ will not affect HER mapping. For each t whose top-K HER matches includes v_0, we update $C_{t_G[B]}$ by adding the last vertex label of $h(\rho_B)$ and call the ranking model $\mathcal{M}_{\text{rank}}$ to get the new top-ranked B-value for the enriched tuple t_G of t.

(2) **[C2] When h is a path match of ρ_A where $A \in \bar{A}$.** Since $h(\rho_A)$ corresponds to an attribute $A \in \bar{A}$ for HER mapping, both $\mathcal{V}_t$ and C_t maintained for tuples t in D may be updated, due to the topological changes, *e.g.,* $h(\rho_A)$ may "promote" v_0 to be a new top HER match for t or "demote" v_0 if v_0 is a current top-K HER match. We re-compute C_t and $\mathcal{V}_t$. If $\mathcal{V}_t$ is changed, we update indices accordingly, and re-populate all $\bar{B}$-attribute values of the enriched tuple t_G of t, by constructing the new candidate sets based on new $\mathcal{V}_t$.

Example 9: Consider ΔD that inserts a new tuple t_6 into D and ΔG that inserts a new edge $e = (v_2, v_{28})$ into G, where $L(e)$ = age and $L(v_{28}) = 38$. We visualize the insertion of e in Figure 4. Given $\rho_1 = (x_0, \text{age}, x_1)$ in Figure 2, $h : \{(v_2, v_{28}) \mapsto (x_0, x_1)\}$ is a path match of ρ_1. Thus, we add the pivot v_2 to Piv(e) and get $P_{v_2} = \{h(\rho_1)\}$. Since $h(\rho_1)$ is a path match of Case **[C1]**, v_2 is still an HER match of t_2 and we populate the age of the enriched tuple of t_2 by $L(v_{28})$, *i.e.,* we update it from null to 38. For ΔD, we simply run BEnrich(ΔD, $G \oplus \Delta G$) to populate the $\bar{B}$-attributes of the enriched tuple of t_6. □

Unit deletion. Unit deletion is processed similarly. We first retrieve the set P_{v_0} of all path matches that are pivoted at v_0 and are removed due to the deletion of e. We process each path match $h \in P_{v_0}$: (1) **[C1] h is a path match of ρ_B.** For each t whose top-K HER matches includes v_0, we update $C_{t_G[B]}$ by removing the value added by $h(\rho_B)$, and update the assignment of $t_G[B]$ based on the ranking model. If $C_{t_G[B]}$ becomes empty, we set $t_G[B]$ = null. (2) **[C2] h is a path match of ρ_A.** We update the sets $\mathcal{V}_t$ and C_t as stated before. If $\mathcal{V}_t$ is updated, we re-populate the $\bar{B}$-attribute values accordingly.

Table 2: Datasets and knowledge graphs

Datasets	$\lvert D\rvert$	$\lvert\bar{A}\rvert$	G	$\lvert V\rvert$	$\lvert E\rvert$
Shoes [86]	3162	3	Wikidata [4]	1.1M	6.3M
Amazon [86]	4589	3	Wikidata [4]	1.1M	6.3M
Person [7]	2.7M	3	Wikidata [4]	1.1M	6.3M
IMDB [2]	2.0M	3	Movie [2]	6.1M	30.0M
Company [86, 95]	28,200	1	Wikidata [4]	1.1M	6.3M
All-xlarge [86, 124]	14,115	3	Wikidata [4]	1.1M	6.3M

Batch updates. Based on unit updates, we develop IncEnrich in Figure 5, for incremental enrichment in response to batch updates $\Delta G = (\Delta G^+, \Delta G^-)$, where ΔG^+ (resp. ΔG^-) is the set of edge insertions (resp. deletions). IncEnrich first retrieves the set P of affected path matches, using Piv[e] for all $e \in \Delta G$ (line 1). Then it groups the affected path matches by pivots [53], so that each path match appears only once even when it has multiple updates (line 2). With a slight abuse of notation, we also denote the group of affected path matches pivoted at v_0 by P_{v_0}. It processes each path match h in P_{v_0} as follows (lines 3 - 11). If h is a path match of ρ_B where $B \in \bar{B}$ (lines 4-6), we check each t whose top-K HER matches include v_0, and update the B-value of the enriched tuple t_G of t if needed. If h is a path match of ρ_A where $A \in \bar{A}$ (lines 7-11), we retrieve the tuples t such that v_0 is in $\mathcal{V}_t$ (resp. C_t) and update $\mathcal{V}_t$ (resp. C_t) if v_0 is no longer a top-K HER match (resp. a candidate match) of t. If $\mathcal{V}_t$ is changed, we re-populate all $\bar{B}$-values of t_G based on the new candidate values from $\mathcal{V}_t$. Finally, the updates (*i.e.,* ΔD_G) are returned (line 12).

Complexity. IncEnrich takes $O(c_{\text{up}}\#\text{Aff}|\Delta G||P_e|)$ time, where #Aff is the maximum number of tuples in D affected by ΔG from one path match, c_{up} is the update cost for one tuple t and P_e is the set of affected path matches for one $e \in \Delta G$, since it takes $O(c_{\text{up}}\#\text{Aff})$ time to process each affected path match in IncEnrich. Thus IncEnrich is in PTIME. This completes the proof of part (2) of Theorem 1.

6 EXPERIMENTAL STUDY

Using benchmarks and real-life data, we empirically evaluated (1) the effectiveness of schema enrichment (SE) and the impact of our policy function on accuracy, (2) the efficiency of SE and (3) the scalability of batch enrichment (BE) and incremental enrichment (IE).

Experimental settings. We start with our experimental settings.

Datasets. We used four benchmarks (two hardest ones with the lowest F_1 of $\mathcal{A}_{\text{ER}}$ and two with the largest F_1 among all benchmarks), and two real-life datasets D. Table 2 reports the statistic and the KG G for each dataset. Here (1) Shoes [86, 95] is an ER benchmark from WDC Product. (2) Amazon [86] is a benchmark of product data. (3) Company is among the largest ER benchmark in [86, 95], of textual data. (4) All-xlarge [86, 124] is the largest ER benchmark from WDC Product with four datasets Computers, Cameras, Watches and Shoes. For the four benchmarks, we used all their attributes as $\bar{A}$.

We also used real-life (5) Person [7] from Wikipedia, where $\bar{A}$ = (name, gender, achieve), and (6) IMDB [2], a dataset of movies and TV Series from 1905 to 2022, where $\bar{A}$ = (title, actor, actress). We set their ground truth by retrieving candidate matching pairs, via Jaccard similarity. We automatically labeled a few tuple pairs with handcrafted rules, and then manually labeled the remaining ones; we also exploited unique links within tuples of Person and IMDB for automatic labeling of a few matches [52]. Finally in total only 1.6K and 1.7K tuples were manually labeled for the two, respectively.

We adopted the same setting of [86], by splitting data to training data S, validation data T and testing data U with the ratio of 3:1:1.

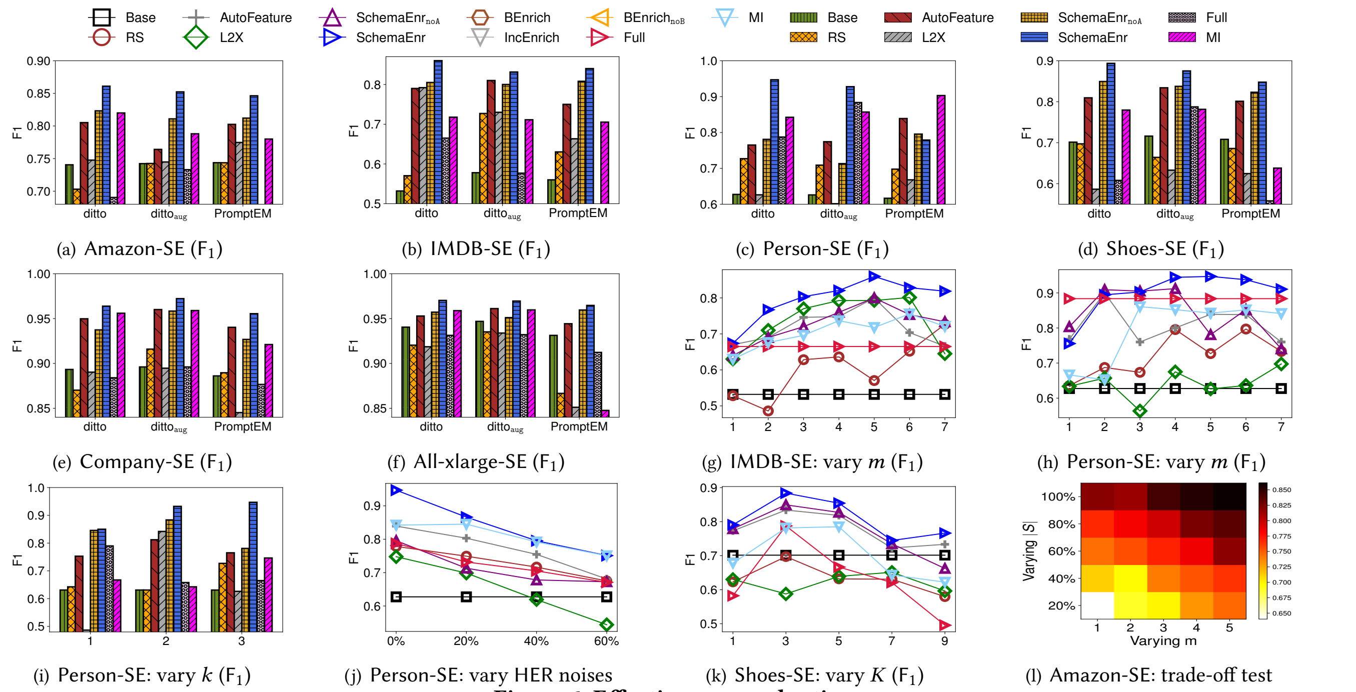


Figure 6: Effectiveness evaluation

ER model $\mathcal{A}_{\text{ER}}$. We used three deep learning $\mathcal{A}_{\text{ER}}$: (1) Ditto [86], a state-of-the-art pre-trained language model. We used RoBerta [89] for Ditto without data augmentation. (2) $\text{Ditto}_{\text{aug}}$ [86], Ditto with data augmentation. (3) PromptEM [124], a state-of-the-art ER model that adopts prompt tuning to fine-tune pre-trained language model. We adopted RoBerta [89] following [124]. The default $\mathcal{A}_{\text{ER}}$ is Ditto. Here Shoes, Amazon, Person, IMDB, Company, All-xlarge, Person and IMDB have 2,063, 6,874, 112,632, 214,736, 20,000 and 20,000 training tuple pairs, respectively. Note that the first four are open-sourced benchmarks that do not require manual annotation.

Hyper-parameters. We adopted CNN with a fully connected layer of 128 dimension for π_θ. The learning rate is 3e-4. The training (resp. validation) batch size is 64 (resp. 1000) for $\mathcal{A}_{\text{ER}}$ and π_θ. We set $m = 5$ as the maximum number of enriched attributes, $k = 3$ as the maximum length of path patterns, $K = 3$ as the number of HER matches in $\mathcal{V}_t$, $I = 200$ as the batch number. For HER, we used 30K, 30K, 233K, 185K, 30K and 50K tuples from D and paths from G to pre-train SentBert in Shoes, Amazon, Company, All-xlarge, Person and IMDB, respectively, in an unsupervised manner.

Baselines. We implemented SchemaEnr in Python, and BEnrich and IncEnrich in Java. We used the following baselines. (1) Base, an ER baseline that does not enrich schema; it fine-tunes $\mathcal{A}_{\text{ER}}$ in instance S of $R = (\bar{A})$ and tests $\mathcal{A}_{\text{ER}}$ in instance U of R. (2) RS, a sampling method that randomly selects m paths from G, *i.e.,* schema $R = (\bar{A})$ is enriched with m new attributes. (3) Full, an ER baseline that enriches schema R with all extractable features/paths from G; since $\mathcal{A}_{\text{ER}}$ only allows at most 512 tokens as input [86], we truncated the enriched features to the maximum size. (4) MI [30], a heuristic method that greedily selects m paths from G as the enriched attributes to maximize the mutual information. (5) AutoFeature [88], a feature augmentation method that selects features from data lakes using DQN; we revised it so that it could select paths from KGs. (6) L2X [35], a feature selection method that adopts mutual information and Gumbel-softmax. For all SE methods (except Base), $\mathcal{A}_{\text{ER}}$ is fine-tuned and evaluated in the enriched training and testing sets.

We also tested the following variants: (7) $\text{SchemaEnr}_{\text{noA}}$, which separately learns $\mathcal{A}_{\text{ER}}$ and then trains π_θ. (8) $\text{SchemaEnr}_{\text{k=1}}$, which only considers paths of length 1 from G as features for enrichment, *i.e.,* $k = 1$. (9) $\text{BEnrich}_{\text{noB}}$, which uses the brute-force HER, such that for each t in D, all vertices in G that share at least one *non-frequent* token with t are taken as HER matches of t. For a fair comparison, we use the same HER method for all baselines whenever possible.

Updates. In IE, we randomly deleted and inserted tuples of D as ΔD, where the inserted tuples are existing ones in D by replacing a few attribute values. Similarly, we constructed ΔG by randomly deleting and inserting edges $e = (v_1, v_2)$ with label l in G, where $v_1, v_2 \in V$ and $l = L(e)$. We set $|\Delta D| = 10\%|D|$ and $|\Delta G| = 10\%|G|$ by default.

Configuration. We conducted the experiments of BE, IE and SE on a single machine powered by 256GB RAM and 32 processors with Intel(R) Xeon(R) Gold 5320 CPU @2.20GHz and Tesla V100 GPUs. Each experiment was run 3 times, and the average is reported here.

Exp-1: Effectiveness We evaluated SchemaEnr in terms of (1) the accuracy F_1 of $\mathcal{A}_{\text{ER}}$; (2) the impact of increasing attributes and lengths of paths; and (3) the impact of HER (see more in [10]).

Accuracy vs. baselines. We tested SchemaEnr in Figure 6(a)-6(f).

(1) SchemaEnr is 4.6% and 5.8% more accurate than $\text{SchemaEnr}_{\text{noA}}$ and $\text{SchemaEnr}_{\text{k=1}}$ (not shown) on average. This shows the need for the joint training strategy and exploration of multi-hop paths from G in SchemaEnr. Learning $\mathcal{A}_{\text{ER}}$ with only $\bar{A}$ does not generalize well to enriched data of schema $(\bar{A}, \bar{B})$, and joint training of $\mathcal{A}_{\text{ER}}$ and π_θ rectifies this. SchemaEnr searches longer paths in G and is able to fetch more informative features than $\text{SchemaEnr}_{\text{k=1}}$.

(2) SchemaEnr consistently beats Base, Full and RS by 15.4%, 19% and 13.4% on average, up to 33%, 65% and 29%, respectively. (a) The results indicate that adding more *useful* contextual information to ER models could increase its accuracy. Note that Ditto and PromptEM are not very accurate on Person and IMDB because the datasets miss some critical attributes for these ER models to distinguish two entities. In contrast, SchemaEnr does better by enriching the schemas with attributes that are not "pseudo-keys" but carry distinguishable values for the entities (see a case study shortly). (b) The learned policy π_θ can find better paths from G ($\bar{B}$-attributes) than random selection. (c) Full does not perform very well, *e.g.,* its F_1 is 20% lower than Base in Amazon. This is because some paths yield low-quality features or null values, leading to the degradation of $\mathcal{A}_{\mathsf{ER}}$. (d) For a similar reason, RS does not always outperform Base, *e.g.,* the F_1 of RS (resp. Base) is 0.49 (resp. 0.53) on IMDB when $m = 2$. (e) Although Base already performs very well on Company and All-xlarge, achieving 0.89 and 0.94 F_1 on average, respectively, SchemaEnr could still leverage knowledge graphs to identify distinguishable features for further improving up to 7.6% F_1 of $\mathcal{A}_{\mathsf{ER}}$.

(3) SchemaEnr consistently outperforms MI, AutoFeature and L2X, *e.g.,* its F_1 is 7.1%, 5.2% and 14.6% higher than the baselines on average, respectively. This is because SchemaEnr finds high-quality paths to improve the ER model $\mathcal{A}_{\mathsf{ER}}$ with the warm-up and mask strategies, while AutoFeature is not designed for path selection and it often misses distinguishing attributes, and MI selects each feature independently, leading to redundant and misleading features. Although L2X selects features for $\mathcal{A}_{\mathsf{ER}}$, it employs Gumbel-softmax to select paths in a single step, yielding indistinguishable attributes in most cases (see a case study shortly). Moreover, SchemaEnr is able to support any $\mathcal{A}_{\mathsf{ER}}$ while L2X requires $\mathcal{A}_{\mathsf{ER}}$ to be differentiable.

Varying m. We varied $m = |\bar{B}|$ from 1 to 7 in Figures 6(g)-6(h). As m increases, SchemaEnr initially gets more accurate, *e.g.,* its F_1 increases from 0.674 to 0.860 on IMDB when m is from 1 to 5. Hence it is able to improve the downstream $\mathcal{A}_{\mathsf{ER}}$ by adding distinguishing attributes from G. However, its F_1 drops as m continues to increase, *e.g.,* it reduces to 0.819 when $m = 7$ on IMDB, because when m reaches, *e.g.,* 5 on IMDB, the enriched attributes are enough to learn $\mathcal{A}_{\mathsf{ER}}$ well, and further increasing m no longer improves F_1; it even reduces F_1, since there are more "noisy" features, *i.e.,* meaningless paths in the search space when m is too large, *e.g.,* 7. In contrast, the F_1 of the baselines may fluctuate, especially when m is large.

Varying k. As shown in Figure 6(i), we varied k from 1 to 3. The F_1 of SchemaEnr increases when k gets larger, *e.g.,* 0.84 to 0.95. Although the ratio of null values slightly increases as k increases, *e.g.,* 35%, 38% and 39% for $k = 1$, 2 and 3, respectively, SchemaEnr is flexible enough to select suitable paths in G and it becomes more accurate. This verifies the need for a reasonably large k, *e.g.,* $k = 3$. SchemaEnr is 12% more accurate than the best of the baselines on average, up to 18%. This verifies that SchemaEnr is able to find distinguishing attributes from G and still has relatively high accuracy in a large search space. AutoFeature, the best baseline, fails to find 3-hop paths because it cannot extract fine-grained paths in graphs.

Impact of HER. We tested the accuracy of our HER method (Section 4.1), defined as the ratio of matched and mismatched tuple-vertex pairs correctly identified to all pairs identified. Since there is no ground truth for HER matches, we sampled a subset of tuples, and used Jaccard similarity to retrieve the top-K vertices in G for each sampled tuple. We manually labeled these pairs as either match or mismatch, 2,545 in total. The results show that on average the HER accuracy is 0.94, and matches are correctly identified in most cases.

Table 3: Case study on Person **for** $m = 5$ **and** $k = 3$

Method	$\bar{B}$	**Path pattern** (where variables are omitted)	MF_1	$\Delta\mathsf{F}_1$
SchemaEnr	B_1	ρ_1=(place-of-birth)	+27%	+27%
	B_2	ρ_2=(place-of-birth, country)	+2%	+29%
	B_3	ρ_3=(place-of-birth, located-in-territorial-entity)	+1%	+30%
	B_4	ρ_4=(languages, has-grammatical-mood)	+1%	+31%
	B_5	ρ_5=(country-of-citizenship, language-used)	+1%	+32%
MI	B_1	ρ_6=(country-of-citizenship, contains-territorial-entry)	-5%	-5%
	B_2	ρ_7=(country-of-citizenship, diplomatic-relation)	+7%	+2%
	B_3	ρ_3=(place-of-birth, located-in-territorial-entity)	+20%	+22%
	B_4	ρ_8=(country-of-citizenship, diplomatic-relation, language-used)	-1%	+21%
	B_5	ρ_{15}=(country-of-citizenship, capital, twinned-admin-body)	+2%	+23%
AutoFeature	B_1	ρ_3=(place-of-birth, located-in-territorial-entity)	+3%	+3%
	B_2	ρ_9=(country-of-citizenship, category-for-people-died-here)	+20%	+23%
L2X	B_1	ρ_{10}=(publisher)	-0.1%	-0.1%
	B_2	ρ_{11}=(partner-in-business-sport)	-0.1%	-0.2%
	B_3	ρ_{12}=(significant-person)	+2.2%	+2%
	B_4	ρ_{13}=(country-for-sport)	+2%	+4%
	B_5	ρ_{14}=(topic-main-template)	+0%	+4%

To further test the impact of HER, we introduce a noise parameter $\beta\%$. We randomly selected $\beta\%$ of tuples, replaced their top-K matches by mismatched vertices, and enriched from the mismatches. As expected, SchemaEnr gets less accurate when more noises present, since it is hard for "wrong" vertices to provide correct features for improving $\mathcal{A}_{\mathsf{ER}}$. When $\beta\% = 60\%$, the accuracy of SchemaEnr drops to 0.751. This justifies the need for accurate HER.

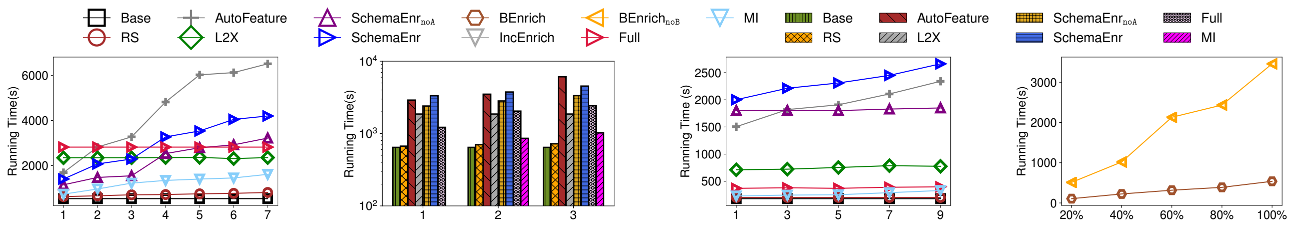
Varying K. Varying K from 1 to 9 in Figure 6(k), SchemaEnr gets higher F_1 when K increases from 1 to 3, since initially a larger K allows SchemaEnr to find good and diverse features. However, when K exceeds a large value, *e.g.,* 5, SchemaEnr performs worse because more noises are involved, increasing the difficulty to learn π_θ.

Trade-off test. We reported the trade-off between the number m of enriched attributes and the size of training set S in SchemaEnr, using a heatmap in Figure 6(l), varying $|S|$ from 20% to 100% and m from 1 to 5. The F_1 in each setting is visualized by the color of a cell in heatmap, where a higher F_1 is shown darker. To achieve similar accuracy (*i.e.,* similar colors), SchemaEnr needs less training data when more distinguishing attributes are enriched, *e.g.,* to make F_1 around 0.75, we need 60% training tuples when the tuples are enriched with 1 attributes, as opposed to 20% training tuples for $m = 5$. In other words, by enriching tuples with 4 more attributes, we save 40% training data for $\mathcal{A}_{\mathsf{ER}}$, maintaining similar accuracy.

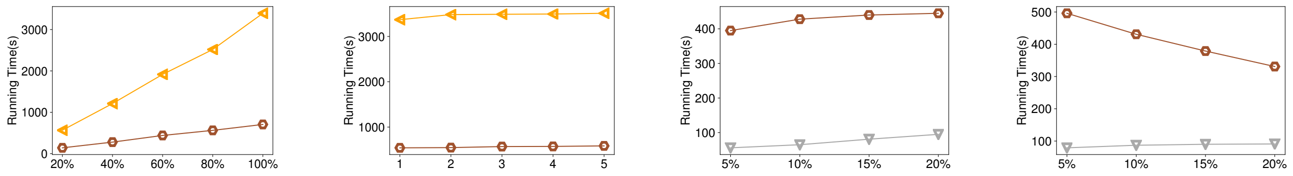
Case study. We showcased attributes $\bar{B}$ enriched from each method for $m = 5$ and $k = 3$ on Person in Table 3, where each B_i ($i \in [1, 5]$) is accompanied with its path pattern ρ (also referred as attributes for simplicity), the accuracy improvement $\Delta\mathsf{F}_1$ of $\mathcal{A}_{\mathsf{ER}}$ (see Section 3.1) when the first i attributes are enriched, and the *marginal* accuracy improvement of $\mathcal{A}_{\mathsf{ER}}$, denoted by MF_1, when B_i itself is enriched.

(1) The attributes enriched by SchemaEnr are distinguishable, *e.g.,* ρ_1 alone is able to improve the F_1 by 27%. Although the marginal improvement of ρ_2-ρ_4 is less than ρ_1, this is reasonable since even 1% of improvement on F_1 is hard when the accuracy is high enough. In contrast, most attributes from other baselines have small and even negative impact on $\mathcal{A}_{\mathsf{ER}}$, *e.g.,* ρ_6 from MI reduces the F_1 by 5%.

(2) Note that ρ_4 and ρ_5 are not pseudo-keys for persons, *e.g.,* two persons can both be citizens from English-speaking countries. However,



(a) Person-SE: vary m (Time) (b) IMDB-SE: vary k (Time) (c) Shoes-SE: vary K (Time) (d) Person-BE: vary $|D|$ (Time)



(e) IMDB-BE: vary $|G|$ (Time) (f) Person-BE: vary m (Time) (g) IMDB-IE: vary $|\Delta D|$ (Time) (h) Person-IE: vary $|\Delta G|$ (Time)

Figure 7: Efficiency and scalability

they are useful for distinguishing persons, *e.g.,* if two persons are from countries with different languages, they are often mismatched.

(3) The baselines miss good attributes. (a) MI introduces redundant or misleading features that hamper the accuracy of $\mathcal{A}_{\text{ER}}$ (*e.g.,* ρ_6 and ρ_8), since it does not consider attributes enriched previously when selecting the next attribute. (b) AutoFeature tends to explore unseen paths in G, but may overlook simple but distinguishable attributes, *e.g.,* ρ_1. Besides, it often misses complicated combinations of paths and thus only finds two attributes. (c) L2X adds all attributes in one step, yielding attributes that are indistinguishable, *e.g.,* ρ_{10}.

Exp-2 Efficiency. We evaluated the (training and inference) time.

Varying m. In Figure 7(a) when varying m from 1 to 7, SchemaEnr takes longer since its search space expands with m, *e.g.,* from 1402s to 3272s when m is from 1 to 4 on Person. This justifies the need of budget m for enrichment. SchemaEnr is not the fastest learner, *e.g.,* it is 1.27X slower than L2X on average. This is because SchemaEnr simultaneously learns $\mathcal{A}_{\text{ER}}$ and π_θ, and its reinforcement learning needs to explore plenty of paths in G to be accurate. This said, the gap between the two is not very large, and SchemaEnr is more accurate than L2X. SchemaEnr is slower than RS, Full and MI, which are based on simple heuristic with the price of lower accuracy.

Varying k. In Figures 7(b), we varied k from 1 to 3. Similar to m, the running time of SchemaEnr increases as k gets larger, as expected, *e.g.,* it takes from 3331s to 4531s when k is from 1 to 3 in IMDB. Although the search space grows exponentially, SchemaEnr does not get much slower due to the mask strategy, *e.g.,* the runtime of $k = 2$ is only 1.1X slower than that of $k = 1$. Considering the significant improvement of F_1, the need for $k > 1$ is justified. We find that when $k = 3$, it suffices to find sensible matches; this echoes the finding of [21, 75] that longer paths hold weaker associations.

Varying K. We varied K from 1 to 9 in Figure 7(c). As expected, as K increases, the running time of SchemaEnr increases, *e.g.,* it takes from 2,002s to 2,663s when K is from 1 to 9. Nevertheless, it is not much slower, indicating that SchemaEnr is able to handle large K.

Joint training. We also revised SchemaEnr by iteratively training π_θ and $\mathcal{A}_{\text{ER}}$ separately, and compared it with the joint training strategy (Figure 3) in IMDB and Person. Joint training is 2.45X faster than iteratively training on average; this justifies the need for joint training to speedup the schema enrichment process.

Exp-3 Scalability. When the enriched schema is in place, we compared the efficiency of BEnrich vs. BEnrich$_{\text{noB}}$ for batch enrichment, and IncEnrich vs. BEnrich for incremental enrichment.

Varying $|D|$. We varied the dataset size $|D|$ from 20% to 100%, and compared BEnrich and BEnrich$_{\text{noB}}$ in Figure 7(d). Both take longer with larger D because they need to enrich more tuples from knowledge graphs. Nonetheless, BEnrich is 6.07X faster than BEnrich$_{\text{noB}}$ on average, which verifies the need for efficient HER methods.

Varying $|G|$. As shown in Figure 7(e) by varying $|G|$ from 20% to 100%, the runtime of all methods increases when $|G|$ gets larger, *e.g.,* BEnrich takes 141s and 563s when $|G|$ is 20% and 80%, respectively. BEnrich is still 5.94X faster than the baseline on average.

Varying m. Varying m from 1 to 5 in Figure 7(f), BEnrich gets slightly slower with larger m; similarly when varying path length k (not shown); *i.e.,* BEnrich is not very sensitive to m and k.

Varying $|\Delta D|$. Fixing $|\Delta G|$ = 10% and varying $|\Delta D|$, we show the runtime of IncEnrich and BEnrich in Figure 7(g). IncEnrich constantly beats its batch counterpart BEnrich. On average IncEnrich is 5.9X faster than BEnrich, since it enriches only tuples in ΔD, not the entire D. Note that it is more costly to handle ΔG (= 10%) than ΔD (= 10%) , since $|G|$ is much larger than $|D|$ in IMDB.

Varying $|\Delta G|$. Fixing $|\Delta D|$ = 10%, we varied the number of edge updates $|\Delta G|$ to G in Figure 7(h). IncEnrich beats BEnrich by 4.77X on average when ΔG varies from 5% to 20%, and by 6.28X when $|\Delta G| = 5\%|G|$. It is faster than BEnrich even when ΔG is up to 20% of Person and IMDB (not shown). This shows the effectiveness of incremental enrichment that focuses on affected paths.

Summary. We find the following. (1) SchemaEnr (schema enrichment) improves the accuracy of ER, *e.g.,* its F_1 increases from 0.674 to 0.86 on IMDB with 4 more attributes. (2) It consistently outperforms the baselines, *e.g.,* on average it is 7.1%, 5.2% and 14.6% more accurate than MI, AutoFeature and L2X, respectively. (3) It beats all its variants, verifying *e.g.,* the benefit of joint training vs. SchemaEnr$_{\text{noA}}$. (4) Our policy π_θ is robust and finds distinguishing attributes from G. (5) Data (batch, incremental) enrichment scales well with different parameters, *e.g.,* BEnrich is 5.94X faster than the baselines on IMDB when $K = 3$ and it is only 1.1X slower when m varies from 1 to 5. (6) Incremental IncEnrich constantly beats the batch one, *e.g.,* when $|\Delta G| = 5\%|G|$, it is 6.28X faster than BEnrich.

7 RELATED WORK

Feature augmentation. Prior work can be classified as follows. (1) *Join-based.* [44, 69, 82, 106, 109, 138] enrich tables by joining external tables in data lakes. (2) *Table discovery and union search.* PEXSEO [44] proposes a framework for joinable table discovery via similarity join. Josie [138] designs an overlap set similarity method to find joinable tables. [135] discovers related tables in a human-in-the-loop manner. COCOA [46] adopts a light-weight index to accelerate tabular enrichment. [76] and [97] find tables that are unionable in data lakes based on the semantics of metadata or attribute correlations. Starmie [47] finds unionable tables via contrastive learning to train column encoders. (3) *Knowledge based*, to enrich tabular data with KGs [42, 57, 63, 94], text [63, 64], information space [43], data warehouse [22], Web data [59] and rule injection [86]. (4) *ML based methods.* AugDiff [110] proposes a diffusion-based framework for multiple instance learning. [36] adopts the auto-encoder neural network to transform raw images with augmented features. [136] proposes spectral feature augmentation to boost contrastive learning. [85] adopts transformation functions for augmentation. (5) *Model-aware methods*, for optimizing downstream models, *e.g.,* AutoFeature [88] applies multi-armed bandit and DQN strategies to get useful features for ML models, [123] selects coresets to make ML feature rich without materialization, ARDA [37] extends datasets by joining correlated tables via coreset and feature selection, Leva [137] transforms tables to graphs and learns embeddings to improve downstream tasks, and [82, 109] explore key-foreign key joins on ML classifiers.

This work differs from the prior work in the following. (1) While some existing methods also incorporate knowledge (*e.g.,* KGs [57, 63, 94]) for feature augmentation, they are not application-aware, *i.e.,* these methods are not designated to improve the performance of a specific type of downstream tasks, while we enrich incomplete schema with bounded attributes to maximize the accuracy of ER. (2) Although [37, 82, 88, 123, 137] optimize for downstream models, they target routine models, not black-box ER models, and focus on finding coarse-grained joinable tables in data lakes, whose schemas are already in place. When adapted to schema enrichment, they encounter issues such as the exponential number of paths, lack of support of textual data, high costs, and outputs that cannot be accepted by $\mathcal{A}_{\mathsf{ER}}$. In contrast, we extract additional fine-grained attributes via paths from KGs to improve ER. This requires us to (a) construct proper attributes from the exponential edge combinations for composing paths, and (b) jointly train the policy and the ER method to be robust to different distributions of the enriched data.

Feature selection. Also related are prior methods for feature selection, classified as follows. (1) *Filter methods,* which rank features based on, *e.g.,* correlation criteria [62], mutual information [30, 35, 78], relief [121], markov blanket [73, 140], etc. Filter methods are fast and model-agnostic, but their features are selected independently. (2) *Wrapper methods,* which search a suboptimal subset of features so that models have the best validation performance, *e.g.,* sequential selection methods [20, 115] and evolutionary algorithms [93, 118, 120]. Such methods find better optimized features, while they incur large cost for exploring the feature space. (3) *Embedded methods,* which embed feature selection into the learning of downstream ML models, where regularization strategies are widely adopted, including LASSO [119], Ridge [67] and Elastic Net [139]. As a trade-off between filter and wrapper methods, embedded methods could find a fairly good subset of features in a short time.

Our work differs from feature selection methods as follows. (1) We aim at improving the accuracy of black-box ER models. (2) While existing methods focus on selecting a subset of given features from a given collection of features, we have to discover features and find good paths in knowledge graphs for composing attributes, via reinforcement learning. (3) We propose three criteria for measuring the paths, namely, diversity, completeness and distinguishability.

Missing values imputation. Imputation methods are proposed to utilize knowledge. (1) *Internal knowledge,* to impute values using rules, *e.g.,* FDs, CFDs [50], DCs [25], PFDs [100] and REEs [55, 56]; ML models, *e.g.,* Baran [90], HoloClean [105, 128], PClean [83] and Restore [66] for relational tables, ORBITS [77] and DeepMVI [29] for time series, and GAIN [129] and GINN [117] for images. (2) *Master data.* [40, 49, 54] correct errors in relations by referencing master data and [55] adopts the chase to correct errors. (3) *Knowledge graph.* FROG [101] proposes imputation methods with complex semantics from knowledge graphs. HER methods, *e.g.,* JedAI [98], parametric simulation [51], Silk [72] and MAGNN [58], link tuples in relations to vertices in graph. Other entity linking methods, *e.g.,* [87, 103], also exist (see [108] for a survey). (4) *Large language models.* [96] uses GPT-3 to impute missing values by proper prompt templates.

While the prior work focuses on missing values for a given schema, we impute incomplete schema, and propose joint training and reinforcement learning for it. For data enrichment, we support both batch and incremental modes, with the parallel scalability. ENRICH supports various HER methods for tuple-vertex matching.

ER. There are plenty of ML-based ER models (surveyed in [38]), which adopt neural networks, attention, RNN and language models. All can be plugged into our scheme as downstream ER models.

8 CONCLUSION

The work is novel in that it (1) studies a new problem of relation enrichment, and settles the complexity of schema enrichment and data (batch, incremental) enrichment; (2) proposes a method to enrich schema by reinforcement learning of a robust policy, data extraction from knowledge graphs, and joint training of the policy and ER models; and (3) develops algorithms for (incremental) enrichment, with the parallel scalability. Our experimental study has verified that the method is promising in improving ER accuracy.

One topic for future work is to collectively enrich multiple relations beyond a single relation. Another topic is to extend ENRICH for improving the accuracy and fairness of ML models beyond ER.

ACKNOWLEDGMENTS

This work was supported by NSFC 62202313, Guangdong Basic and Applied Basic Research Foundation 2022A1515010120, Longhua Science and Technology Innovation Bureau 10162A20220720B12AB12.

REFERENCES

[1] 2017. Identity fraud's impact on the insurance sector. https://legal.thomsonreuters.com/en/insights/articles/identity-frauds-impact-on-the-insurance-sector.

[2] 2019. IMDB. *https://www.imdb.com/interfaces/*.

[3] 2020. Knowledge Graphs for Financial Services. https://www2.deloitte.com/content/dam/Deloitte/nl/Documents/risk/deloitte-nl-risk-knowledge-graphs-financial-services.pdf.

[4] 2022. DBpedia. *http://wiki.dbpedia.org*.

[5] 2022. Fraud detection using knowledge graph: How to detect and visualize fraudulent activities. https://www.nebula-graph.io/posts/fraud-detection-using-knowledge-and-graph-database.

[6] 2022. How Fraudsters Create Fake Identities. https://www.shift-technology.com/resources/perspectives/sme-perspectives-how-fraudsters-create-fake-identities.

[7] 2022. Wikemedia. https://www.kaggle.com/datasets/kenshoresearch/kensho-derived-wikimedia-data.

[8] 2022. Wikidata – Recent changes. *https://www.amazon.science/blog/combining-knowledge-graphs-quickly-and-accurately*.

[9] 2022. Wikipedia. *https://www.wikipedia.org*.

[10] 2023. Code, datasets and full version. https://github.com/SICS-Fundamental-Research-Center/Enrichment.

[11] 2023. IMDb Non-Commercial Datasets. https://developer.imdb.com/non-commercial-datasets.

[12] 2023. Leverage Data Enrichment to Ensure You're Dealing with Real People. https://seon.io/resources/online-insurance-fraud/.

[13] 2023. SEON. https://seon.io/.

[14] 2023. SIFT. https://sift.com/.

[15] 2023. Social Network Usage and Growth Statistics. *https://backlinko.com/social-media-users*.

[16] 2023. STARBUCKS eGIFT. https://www.starbucks.com/terms/gift-card-offer-terms/.

[17] 2023. Wikidata:WikiProject Disambiguation pages. https://www.wikidata.org/wiki/Wikidata:WikiProject_Disambiguation_pages.

[18] Aisha Abdallah, Mohd Aizaini Maarof, and Anazida Zainal. 2016. Fraud detection system: A survey. *Journal of Network and Computer Applications* 68 (2016), 90–113.

[19] Ghadeer Abuoda, Saravanan Thirumuruganathan, and Ashraf Aboulnaga. 2022. Accelerating Entity Lookups in Knowledge Graphs Through Embeddings. In *ICDE*. IEEE, 1111–1123.

[20] David W. Aha and Richard L. Bankert. 1995. A Comparative Evaluation of Sequential Feature Selection Algorithms. In *Learning from Data - Fifth International Workshop on Artificial Intelligence and Statistics (AISTATS)*. Springer, 199–206.

[21] Boanerges Aleman-Meza, Christian Halaschek-Wiener, Ismailcem Budak Arpinar, and Amit P. Sheth. 2003. Context-Aware Semantic Association Ranking. In *SWDB*. 33–50.

[22] Rohit Ananthakrishna, Surajit Chaudhuri, and Venkatesh Ganti. 2002. Eliminating Fuzzy Duplicates in Data Warehouses. In *VLDB*. 586–597.

[23] Arvind Arasu, Michaela Götz, and Raghav Kaushik. 2010. On active learning of record matching packages. In *SIGMOD*. 783–794.

[24] Arvind Arasu, Christopher Ré, and Dan Suciu. 2009. Large-Scale Deduplication with Constraints Using Dedupalog. In *ICDE*. 952–963.

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

[26] Abolfazl Asudeh, Nima Shahbazi, Zhongjun Jin, and H. V. Jagadish. 2021. Identifying Insufficient Data Coverage for Ordinal Continuous-Valued Attributes. In *SIGMOD*. 129–141.

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

[28] Zeinab Bahmani, Leopoldo E. Bertossi, and Nikolaos Vasiloglou. 2017. ERBlox: Combining matching dependencies with machine learning for entity resolution. *Int. J. Approx. Reasoning* 83 (2017), 118–141.

[29] Parikshit Bansal, Prathamesh Deshpande, and Sunita Sarawagi. 2021. Missing Value Imputation on Multidimensional Time Series. *PVLDB* 14, 11 (2021), 2533–2545.

[30] Roberto Battiti. 1994. Using mutual information for selecting features in supervised neural net learning. *IEEE Trans. Neural Networks* 5, 4 (1994), 537–550.

[31] Mario Beraha, Alberto Maria Metelli, Matteo Papini, Andrea Tirinzoni, and Marcello Restelli. 2019. Feature Selection via Mutual Information: New Theoretical Insights. In *International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–9.

[32] 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.* 52, 3 (2013), 441–482.

[33] Gunawan Budiprasetyo. 2019. *Optimisation classification on the web of data using linked data. A study case: Movie popularity classification.* Ph.D. Dissertation. University of Southampton.

[34] Gabrielle Karine Canalle, Bernadette Farias Loscio, and Ana Carolina Salgado. 2017. A strategy for selecting relevant attributes for entity resolution in data integration systems. In *International Conference on Enterprise Information Systems*, Vol. 2. SCITEPRESS, 80–88.

[35] Jianbo Chen, Le Song, Martin Wainwright, and Michael Jordan. 2018. Learning to explain: An information-theoretic perspective on model interpretation. In *International Conference on Machine Learning*. PMLR, 883–892.

[36] Zitian Chen, Yanwei Fu, Yinda Zhang, Yu-Gang Jiang, Xiangyang Xue, and Leonid Sigal. 2019. Multi-level semantic feature augmentation for one-shot learning. *IEEE Transactions on Image Processing* 28, 9 (2019), 4594–4605.

[37] Nadiia Chepurko, Ryan Marcus, Emanuel Zgraggen, Raul Castro Fernandez, Tim Kraska, and David R. Karger. 2020. ARDA: Automatic Relational Data Augmentation for Machine Learning. *PVLDB* 13, 9 (2020), 1373–1387.

[38] 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.

[39] E. F. Codd. 1979. Extending the Database Relational Model to Capture More Meaning. *ACM Trans. Database Syst.* 4, 4 (1979), 397–434.

[40] Ting Deng, Wenfei Fan, and Floris Geerts. 2016. Capturing Missing Tuples and Missing Values. *ACM Trans. Database Syst.* 41, 2 (2016), 10:1–10:47.

[41] Ting Deng, Wenfei Fan, Ping Lu, Xiaomeng Luo, Xiaoke Zhu, and Wanhe An. 2022. Deep and Collective Entity Resolution in Parallel. In *ICDE*. IEEE, 2060–2072.

[42] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2020. TURL: Table Understanding through Representation Learning. *PVLDB* (2020).

[43] Xin Dong, Alon Y. Halevy, and Jayant Madhavan. 2005. Reference Reconciliation in Complex Information Spaces. In *SIGMOD*. ACM, 85–96.

[44] Yuyang Dong, Kunihiro Takeoka, Chuan Xiao, and Masafumi Oyamada. 2021. Efficient joinable table discovery in data lakes: A high-dimensional similarity-based approach. In *ICDE*. IEEE, 456–467.

[45] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq R. Joty, Mourad Ouzzani, and Nan Tang. 2018. Distributed Representations of Tuples for Entity Resolution. *PVLDB* 16, 8 (2018), 1944–1957.

[46] Mahdi Esmailoghli, Jorge-Arnulfo Quiané-Ruiz, and Ziawasch Abedjan. 2021. COCOA: COrrelation COefficient-Aware Data Augmentation. In *EDBT*. 331–336.

[47] Grace Fan, Jin Wang, Yuliang Li, Dan Zhang, and Renée J. Miller. 2023. Semantics-aware Dataset Discovery from Data Lakes with Contextualized Column-based Representation Learning. *PVLDB* 16, 7 (2023), 1726–1739.

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

[49] Wenfei Fan and Floris Geerts. 2010. Relative information completeness. *ACM Trans. Database Syst.* 35, 4 (2010), 27:1–27:44.

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

[51] Wenfei Fan, Liang Geng, Ruochun Jin, Ping Lu, Resul Tugey, and Wenyuan Yu. 2022. Linking Entities across Relations and Graphs. In *ICDE*. IEEE, 634–647.

[52] Wenfei Fan, Ziyan Han, Weilong Ren, Ding Wang, Yaoshu Wang, Min Xie, and Mengyi Yan. 2023. Splitting Tuples of Mismatched Entities. *Proc. ACM Manag. Data* (2023).

[53] Wenfei Fan, Chunming Hu, and Chao Tian. 2017. Incremental Graph Computations: Doable and Undoable. In *SIGMOD*. 155–169.

[54] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Wenyuan Yu. 2012. Towards certain fixes with editing rules and master data. *VLDB J.* 21, 2 (2012), 213–238.

[55] Wenfei Fan, Ping Lu, and Chao Tian. 2020. Unifying Logic Rules and Machine Learning for Entity Enhancing. *Sci. China Inf. Sci.* 63, 7 (2020).

[56] Wenfei Fan, Chao Tian, Yanghao Wang, and Qiang Yin. 2021. Parallel Discrepancy Detection and Incremental Detection. *PVLDB* 14, 8 (2021), 1351–1364.

[57] Lior Friedman and Shaul Markovitch. 2018. Recursive feature generation for knowledge-based learning. *arXiv preprint arXiv:1802.00050* (2018).

[58] Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. 2020. MAGNN: Metapath aggregated graph neural network for heterogeneous graph embedding. In *The Web Conference 2020*. 2331–2341.

[59] Sainyam Galhotra, Udayan Khurana, Oktie Hassanzadeh, Kavitha Srinivas, Horst Samulowitz, and Miao Qi. 2019. Automated feature enhancement for predictive modeling using external knowledge. In *International Conference on Data Mining Workshops (ICDMW)*. IEEE, 1094–1097.

[60] Michael Garey and David Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company.

[61] Songtao Guo, Xin Luna Dong, Divesh Srivastava, and Remi Zajac. 2010. Record Linkage with Uniqueness Constraints and Erroneous Values. *PVLDB* 3, 1 (2010), 417–428.

[62] Mark A. Hall. 2000. Correlation-based Feature Selection for Discrete and Nu-

meric Class Machine Learning. In *International Conference on Machine Learning (ICML)*. Morgan Kaufmann, 359–366.

[63] Asaf Harari and Gilad Katz. 2022. Automatic features generation and selection from external sources: A DBpedia use case. *Information Sciences* 582 (2022), 398–414.

[64] Asaf Harari and Gilad Katz. 2022. Few-Shot Tabular Data Enrichment Using Fine-Tuned Transformer Architectures. In *ACL*. Association for Computational Linguistics, 1577–1591.

[65] Qi He, Jaewon Yang, and Baoxu Shi. 2020. Constructing knowledge graph for social networks in a deep and holistic way. In *Companion Proceedings of the Web Conference 2020*. 307–308.

[66] Benjamin Hilprecht and Carsten Binnig. 2021. ReStore - Neural Data Completion for Relational Databases. In *SIGMOD*. 710–722.

[67] Arthur E. Hoerl and Robert W. Kennard. 2000. Ridge Regression: Biased Estimation for Nonorthogonal Problems. *Technometrics* 42, 1 (2000), 80–86.

[68] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutiérrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan F. Sequeda, Steffen Staab, and Antoine Zimmermann. 2021. Knowledge Graphs. *ACM Comput. Surv.* 54, 4 (2021), 71:1–71:37.

[69] Xuming Hu, Shen Wang, Xiao Qin, Chuan Lei, Zhengyuan Shen, Christos Faloutsos, Asterios Katsifodimos, George Karypis, Lijie Wen, and Philip S. Yu. 2023. Automatic Table Union Search with Tabular Representation Learning. In *Findings of the Association for Computational Linguistics: ACL*. Association for Computational Linguistics.

[70] Shengyi Huang and Santiago Ontañón. 2022. A Closer Look at Invalid Action Masking in Policy Gradient Algorithms. In *the Thirty-Fifth International Florida Artificial Intelligence Research Society Conference (FLAIRS)*.

[71] Vassilis N Ioannidis, Xiang Song, Saurav Manchanda, Mufei Li, Xiaoqin Pan, Da Zheng, Xia Ning, Xiangxiang Zeng, and George Karypis. 2020. DRKG-drug repurposing knowledge graph for covid-19. https://github.com/gnn4dr/DRKG/.

[72] Robert Isele, Anja Jentzsch, and Christian Bizer. 2010. Silk server-adding missing links while consuming linked data. In *COLD*. 85–96.

[73] Kashif Javed, Sameen Maruf, and Haroon A. Babri. 2015. A two-stage Markov blanket based feature selection algorithm for text classification. *Neurocomputing* 157 (2015), 91–104.

[74] Jungo Kasai, Kun Qian, Sairam Gurajada, Yunyao Li, and Lucian Popa. 2019. Low-resource Deep Entity Resolution with Transfer and Active Learning. In *ACL*. 5851–5861.

[75] Yoed N Kenett, Effi Levi, David Anaki, and Miriam Faust. 2017. The semantic distance task: Quantifying semantic distance with semantic network path length. *Journal of Experimental Psychology: Learning, Memory, and Cognition* 43, 9 (2017), 1470.

[76] Aamod Khatiwada, Grace Fan, Roee Shraga, Zixuan Chen, Wolfgang Gatterbauer, Renée J. Miller, and Mirek Riedewald. 2023. SANTOS: Relationship-based Semantic Table Union Search. *Proc. ACM Manag. Data* 1, 1 (2023), 9:1–9:25.

[77] Mourad Khayati, Ines Arous, Zakhar Tymchenko, and Philippe Cudré-Mauroux. 2020. ORBITS: Online Recovery of Missing Values in Multiple Time Series Streams. *PVLDB* 14, 3 (2020), 294–306.

[78] Ron Kohavi and George H. John. 1997. Wrappers for Feature Subset Selection. *Artif. Intell.* 97, 1-2 (1997), 273–324.

[79] loannis Koumarelas, Thorsten Papenbrock, and Felix Naumann. 2020. MDedup: Duplicate Detection with Matching Dependencies. *PVLDB* 13, 5 (2020), 712–725.

[80] Walter Kropatsch. 1996. Building irregular pyramids by dual-graph contraction. In *Vision Image and Signal Processing*.

[81] Clyde P. Kruskal, Larry Rudolph, and Marc Snir. 1990. A complexity theory of efficient parallel algorithms. *Theoretical Computer Science* 71, 1 (1990), 95–132.

[82] Arun Kumar, Jeffrey Naughton, Jignesh M Patel, and Xiaojin Zhu. 2016. To join or not to join? Thinking twice about joins before feature selection. In *SIGMOD*. 19–34.

[83] Alexander K. Lew, Monica Agrawal, David A. Sontag, and Vikash Mansinghka. 2021. PClean: Bayesian Data Cleaning at Scale with Domain-Specific Probabilistic Programming. In *International Conference on Artificial Intelligence and Statistics, (AISTATS) (Proceedings of Machine Learning Research)*.

[84] Chenjie Li, Zhengjie Miao, Qitian Zeng, Boris Glavic, and Sudeepa Roy. 2021. Putting things into context: Rich explanations for query answers using join graphs. In *SIGMOD*. 1051–1063.

[85] Pan Li, Da Li, Wei Li, Shaogang Gong, Yanwei Fu, and Timothy M Hospedales. 2021. A simple feature augmentation for domain generalization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 8886–8895.

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

[87] Xueling Lin, Haoyang Li, Hao Xin, Zijian Li, and Lei Chen. 2020. KBPearl: A Knowledge Base Population System Supported by Joint Entity and Relation Linking. *PVLDB* 13, 7 (2020), 1035–1049.

[88] Jiabin Liu, Chengliang Chai, Yuyu Luo, Yin Lou, Jianhua Feng, and Nan Tang. 2022. Feature Augmentation with Reinforcement Learning. In *ICDE*. IEEE, 3360–3372.

[89] 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. *CoRR* abs/1907.11692 (2019).

[90] Mohammad Mahdavi and Ziawasch Abedjan. 2020. Baran: Effective Error Correction via a Unified Context Representation and Transfer Learning. *PVLDB* 13, 11 (2020), 1948–1961.

[91] Andy Maule, Wolfgang Emmerich, and David S Rosenblum. 2008. Impact analysis of database schema changes. In *international conference on Software engineering*. 451–460.

[92] Zhengjie Miao, Yuliang Li, and Xiaolan Wang. 2021. Rotom: A Meta-Learned Data Augmentation Framework for Entity Matching, Data Cleaning, Text Classification, and Beyond. In *SIGMOD*. ACM, 1303–1316.

[93] Alberto Moraglio, Cecilia Di Chio, and Riccardo Poli. 2007. Geometric Particle Swarm Optimisation. In *EuroGP (Lecture Notes in Computer Science, Vol. 4445)*. Springer, 125–136.

[94] Michalis Mountantonakis and Yannis Tzitzikas. 2017. How linked data can aid machine learning-based tasks. In *International Conference on Theory and Practice of Digital Libraries (TPDL)*. 155–168.

[95] 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*. 19–34.

[96] Avanika Narayan, Ines Chami, Laurel J. Orr, and Christopher Ré. 2022. Can Foundation Models Wrangle Your Data? *PVLDB* 16, 4 (2022), 738–746.

[97] Fatemeh Nargesian, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. 2018. Table Union Search on Open Data. *PVLDB* 11, 7 (2018), 813–825.

[98] George Papadakis, George Mandilaras, Luca Gagliardelli, Giovanni Simonini, Emmanouil Thanos, George Giannakopoulos, Sonia Bergamaschi, Themis Palpanas, and Manolis Koubarakis. 2020. Three-dimensional entity resolution with JedAI. *Information Systems* 93 (2020), 101565.

[99] Jan Peters and J. Andrew Bagnell. 2017. Policy Gradient Methods. In *Encyclopedia of Machine Learning and Data Mining*. Springer.

[100] Abdulhakim Qahtan, Nan Tang, Mourad Ouzzani, Yang Cao, and Michael Stonebraker. 2020. Pattern functional dependencies for data cleaning. *PVLDB* 13, 5 (2020), 684–697.

[101] Zhixin Qi, Hongzhi Wang, Jianzhong Li, and Hong Gao. 2018. FROG: Inference from knowledge base for missing value imputation. *Knowl. Based Syst.* 145 (2018), 77–90.

[102] Kun Qian, Lucian Popa, and Prithviraj Sen. 2017. Active Learning for Large-Scale Entity Resolution. In *CIKM*. 1379–1388.

[103] Priya Radhakrishnan, Partha P. Talukdar, and Vasudeva Varma. 2018. ELDEN: Improved Entity Linking Using Densified Knowledge Graphs. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*. Association for Computational Linguistics, 1844–1853.

[104] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Conference on Empirical Methods in Natural Language Processing*.

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

[106] Aécio Santos, Aline Bessa, Fernando Chirigati, Christopher Musco, and Juliana Freire. 2021. Correlation sketches for approximate join-correlation queries. In *SIGMOD*. 1531–1544.

[107] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *CoRR* abs/1707.06347 (2017).

[108] Özge Sevgili, Artem Shelmanov, Mikhail Y. Arkhipov, Alexander Panchenko, and Chris Biemann. 2022. Neural entity linking: A survey of models based on deep learning. *Semantic Web* 13, 3 (2022), 527–570.

[109] Vraj Shah, Arun Kumar, and Xiaojin Zhu. 2017. Are key-foreign key joins safe to avoid when learning high-capacity classifiers? *arXiv preprint arXiv:1704.00485* (2017).

[110] Zhuchen Shao, Liuxi Dai, Yifeng Wang, Haoqian Wang, and Yongbing Zhang. 2023. AugDiff: Diffusion based Feature Augmentation for Multiple Instance Learning in Whole Slide Image. *arXiv preprint arXiv:2303.06371* (2023).

[111] Shubhranshu Shekhar, Deepak Pai, and Sriram Ravindran. 2020. Entity resolution in dynamic heterogeneous networks. In *Companion Proceedings of the Web Conference 2020*. 662–668.

[112] Feichen Shen and Yugyung Lee. 2016. Knowledge discovery from biomedical ontologies in cross domains. *PloS one* 11, 8 (2016), e0160005.

[113] Kai Shu, Suhang Wang, Jiliang Tang, Reza Zafarani, and Huan Liu. 2016. User Identity Linkage across Online Social Networks: A Review. *SIGKDD Explor.* 18, 2 (2016), 5–17.

[114] Dag Sjøberg. 1993. Quantifying schema evolution. *Information and Software Technology* 35, 1 (1993), 35–44.

[115] Petr Somol, Pavel Pudil, Jana Novovicová, and Pavel Paclík. 1999. Adaptive floating search methods in feature selection. *Pattern Recognit. Lett.* 20, 11-13 (1999), 1157–1163.
[116] Shaoxu Song, Yu Sun, Aoqian Zhang, Lei Chen, and Jianmin Wang. 2018. Enriching data imputation under similarity rule constraints. *TKDE* 32, 2 (2018), 275–287.
[117] Indro Spinelli, Simone Scardapane, and Aurelio Uncini. 2020. Missing data imputation with adversarially-trained graph convolutional networks. *Neural Networks* (2020).
[118] El-Ghazali Talbi, Laetitia Jourdan, José García-Nieto, and Enrique Alba. 2008. Comparison of population based metaheuristics for feature selection: Application to microarray data classification. In *International Conference on Computer Systems and Applications (AICCSA)*. IEEE Computer Society, 45–52.
[119] R. Tibshirani. 1996. Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society (Series B)* 58 (1996), 267–288.
[120] Chung-Jui Tu, Li-Yeh Chuang, Jun-Yang Chang, and Cheng-Hong Yang. 2006. Feature Selection using PSO-SVM. In *International MultiConference of Engineers and Computer Scientists (IMECS) (Lecture Notes in Engineering and Computer Science)*. Newswood Limited, 138–143.
[121] Ryan J. Urbanowicz, Melissa Meeker, William G. La Cava, Randal S. Olson, and Jason H. Moore. 2018. Relief-based feature selection: Introduction and review. *J. Biomed. Informatics* 85 (2018), 189–203.
[122] Aäron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation Learning with Contrastive Predictive Coding. *CoRR* abs/1807.03748 (2018).
[123] Jiayi Wang, Chengliang Chai, Nan Tang, Jiabin Liu, and Guoliang Li. 2022. Coresets over Multiple Tables for Feature-rich and Data-efficient Machine Learning. *PVLDB* 16, 1 (2022), 64–76.
[124] Pengfei Wang, Xiaocan Zeng, Lu Chen, Fan Ye, Yuren Mao, Junhao Zhu, and Yunjun Gao. 2022. PromptEM: Prompt-tuning for Low-resource Generalized Entity Matching. *PVLDB* 16, 2 (2022), 369–378.
[125] Yue Wang and Shaofeng Zou. 2022. Policy Gradient Method For Robust Reinforcement Learning. In *International Conference on Machine Learning (ICML)*.
[126] Melanie Weis and Felix Naumann. 2005. DogmatiX Tracks down Duplicates in XML. In *SIGMOD*. ACM, 431–442.
[127] Steven Euijong Whang and Hector Garcia-Molina. 2013. Joint entity resolution on multiple datasets. *VLDB J.* 22, 6 (2013), 773–795.
[128] Richard Wu, Aoqian Zhang, Ihab F. Ilyas, and Theodoros Rekatsinas. 2020. Attention-based Learning for Missing Data Imputation in HoloClean. In *MLSys 2020*.
[129] Jinsung Yoon, James Jordon, and Mihaela van der Schaar. 2018. GAIN: Missing Data Imputation using Generative Adversarial Nets. In *ICML*. PMLR, 5675–5684.
[130] Brit Youngmann, Michael Cafarella, Yuval Moskovitch, and Babak Salimi. 2023. On Explaining Confounding Bias. In *ICDE*. IEEE, 1846–1859.
[131] Brit Youngmann, Michael Cafarella, Babak Salimi, and Anna Zeng. 2023. Causal Data Integration. *PVLDB* 16, 10 (2023), 2659–2665.
[132] Wenhao Yu, Chenguang Zhu, Zaitang Li, Zhiting Hu, Qingyun Wang, Heng Ji, and Meng Jiang. 2022. A Survey of Knowledge-enhanced Text Generation. *ACM Comput. Surv.* 54, 11s (2022), 227:1–227:38.
[133] Reza Zafarani and Huan Liu. 2016. Users joining multiple sites: Friendship and popularity variations across sites. *Inf. Fusion* 28 (2016), 83–89.
[134] Dongxiang Zhang, Long Guo, Xiangnan He, Jie Shao, Sai Wu, and Heng Tao Shen. 2018. A Graph-Theoretic Fusion Framework for Unsupervised Entity Resolution. In *ICDE*.
[135] Yi Zhang and Zachary G Ives. 2020. Finding related tables in data lakes for interactive data science. In *SIGMOD*. ACM, 1951–1966.
[136] Yifei Zhang, Hao Zhu, Zixing Song, Piotr Koniusz, and Irwin King. 2022. Spectral Feature Augmentation for Graph Contrastive Learning and Beyond. *arXiv preprint arXiv:2212.01026* (2022).
[137] Zixuan Zhao and Raul Castro Fernandez. 2022. Leva: Boosting Machine Learning Performance with Relational Embedding Data Augmentation. In *SIGMOD*. ACM, 1504–1517.
[138] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J Miller. 2019. Josie: Overlap set similarity search for finding joinable tables in data lakes. In *SIGMOD*. ACM, 847–864.
[139] H. Zou and T. Hastie. 2003. Regularization and Variable Selection via the Elastic Net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* (2003).
[140] Xiaohan Zuo, Peng Lu, Xi Liu, Yibo Gao, Yiping Yang, and Jianxin Chen. 2011. An improved feature selection algorithm based on Markov blanket. In *International Conference on Biomedical Engineering and Informatics, (BMEI)*. IEEE, 1645–1649.