Enriching Relations with Additional Attributes for ER

Mengyi Yan[1], Wenfei Fan[1,2,3], Yaoshu Wang[2], Min Xie[2]
[1]Beihang University [2]Shenzhen Institute of Computing Sciences [3]University of Edinburgh
yanmy@act.buaa.edu.cn,wenfei@inf.ed.ac.uk,{yaoshuw,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 20.6% (percentage points) by adding 5 attributes, up to 33%; moreover, the scheme is efficient for batch and incremental 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, important attributes are often missing from schema R, and as a consequence, the tuples in D do not have enough information for the application. This happens because either at the time of schema design, the information about those attributes was not available, or the schema was designed for a rather different application. For example, Adobe was originally known for its desktop software products, but later on it transitioned from selling software licenses to a subscription-based model where new information (*e.g.,* collaborative features) from customers is needed. Regardless of the reasons, relation D often misses features needed for our application. That is, we often encounter not only missing data (values and tuples) of a fixed schema, but also missing attributes from the schema, *i.e.,* "incomplete schema". In this paper, we focus on entity resolution (ER) as our target application.

Example 1: Consider Table 1 with 5 tuples t_1-t_5 for 4 entities e_1-e_4. The tuples have a schema with attributes name, gender, nationality and occupation. A not-so-sophisticated ER method $\mathcal{A}_{\text{ER}}$ may predict that t_1 and t_2 are the same person (a false positive, FN), since they have similar names and are both American actors. It may also miss the true match of t_2 and t_3 (a false negative, FN), due to different names and occupations, although actor Mark Wahlberg started out as a rapper in his early life using the stage name "Marky Mark".

Fortunately, additional attributes can help us reduce both FPs and FNs of ER. (a) If an additional born attribute is available, it provides a strong evidence for $\mathcal{A}_{\text{ER}}$ to tell that t_1 and t_2 are not the same person, since they were born in different years. (b) If we further enrich the schema with attribute spouse_name, $\mathcal{A}_{\text{ER}}$ can identify t_2 and t_3, since they are married to the same person. □

tid	name	gender	nationality	occupation	born	spouse_name	eid
t_1	Mark L. Walberg	M	USA	Actor, host	1962	Robbi Morgan	e_1
t_2	Mark Wahlberg	M	USA	Actor	1971	Rhea Durham	e_2
t_3	Marky Mark	M	USA	Rapper	1971	Rhea Durham	e_2
t_4	Robbi Morgan	F	USA	Actress	1961	Mark L. Walberg	e_3
t_5	Rhea Durham	F	USA	Model	1978	Mark Wahlberg	e_4

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

Missing attributes are as damaging as missing data, but the issue has not received much attention. While there has been a host of work on missing data imputation [16, 17, 20, 31, 39, 40, 44, 46, 57, 68, 92, 97, 105, 117, 118], no prior work has systematically studied how to enrich incomplete schema in order to improve ER accuracy. While there have been efforts on feature augmentation, this line of work either does not target downstream applications [35, 49, 55, 73, 76, 85, 98, 100, 101, 123, 125], or is not developed for ER [79, 112].

To enrich a relation D of schema R for ER, we want to extract information from external sources *e.g.,* textual data [55, 56], information space [34], XML data [115], data warehouse [12] and structured Web data [51]. In fact, knowledge enrichment has been practiced in medical knowledge discovery [102], network dynamics analysis [120], e-commerce [7], recommendation [103] and knowledge-enhanced text generation [119]. Among external sources, knowledge graphs (KGs) are particularly promising for ER. KGs often offer the ability to link related entities and disambiguate entities with similar names [8], making it feasible to improve the ER accuracy. Moreover, documents, tables, master data and even outputs of LLMs are increasingly unified into KGs. Better still, KGs are becoming more focused when LLMs are fine-tuned to specific domains and thus, are able to cover knowledge in these domains.

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 in a specific domain. 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 [26], 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 a bounded number of relevant features; this setting is also to accommodate downstream ER models, which often have a bounded input length (*e.g.,* at most 512 tokens for Bert).

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 graph 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 knowledge graph G. We learn a policy via reinforcement learning to extract at most m features of the matching vertices from G, and extend 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 downstream ML 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 [72]. We defer the parallelization to [6] 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 20.6% on average, up to 33%, by adding 5 attributes. (b) It is on average 7% (resp. 18%) more accurate than ML models for feature augmentation (resp. selection), up to 18.2% (resp. 32.6%). (c) Batch (resp. incremental) enrichment is 5.94X (resp. 3.1X) faster than the baselines on average. (4) The incremental method beats the batch one when updates to D and G are up to 20%, and is 5.77X 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 [30], 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 [59], 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

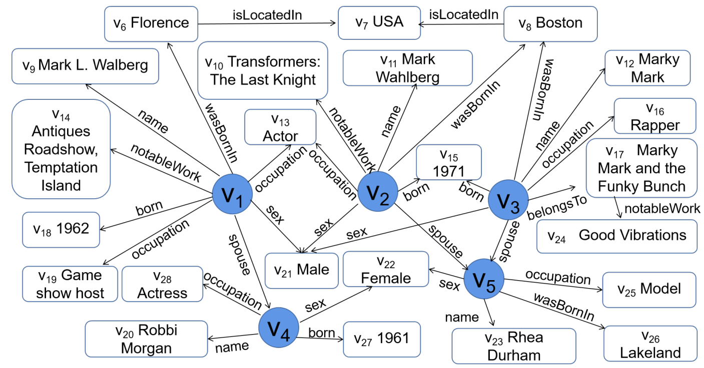


Figure 1: An example knowledge graph

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 models [14, 36, 65, 77, 86, 94, 121], logic rules [15, 18, 23, 38, 53, 70, 116] and hybrid of the two [19, 32, 47]. We focus on ML-based ER methods, which take quadratic-time in the worst case to identify matches (after the relevant ML models are trained).

*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 S of tuples of schema R, computes a set:

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

Here $t \Rightarrow v$ denotes that tuple t and vertex v make a *match*, *i.e.,* t and v refer to the same real-world entity. Note that for each $t \in S$, there can be multiple vertices v in G that match t. We refer to f_{HER} as the HER *mapping* and $f_{\mathsf{HER}}(S, G)$ as the *match relation* of S and G.

Several methods are in place for HER, *e.g.,* rule-based JedAI [90], parametric simulation [41], and ML-based Silk [62], MAGNN [50] and EMBLOOKUP [9]. In particular, it is in $O(|S||G|)$ time to compute all HER matches across S and G by parametric simulation [41].

We assume schema R has enough information for a well-designed HER mapping f_{HER} to map tuples of R to entities in G (see Section 4.1 for our HER mapping). Intuitively, HER helps us extract features from external data sources and enrich a schema with additional attributes for downstream ER; this cannot be achieved by improving ER models alone on datasets of a fixed schema. Moreover, HER is expected to get more accurate when ML techniques evolve.

Example 2: Consider an example knowledge graph G in Figure 1 for the person tuples in Table 1, where t_i matches v_i in G ($i \in [1, 5]$).

HER differs from ER in the following. (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.,* attribute nationality of t_1 vs. path $\rho = (v_1, v_6, v_7)$ in G; (c) not every attribute in t can find a matching path in G, and vice versa, *e.g.,* v_2 has notable work "Transformers", which finds no corresponding attribute in Table 1; and (d) t and v often have different descriptions for the same property, *e.g.,* attribute gender of t_3 vs. edge sex of v_3. □

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 $(\mathsf{id}, A_1, \ldots, A_n)$ of attributes, consider a relation D of R, a KG G, and an ER model $\mathcal{A}_{\mathsf{ER}}$.

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

Following Codd [30], 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 $f(t.\mathsf{id}) = e$, *i.e.,* for each tuple t in D, $f(t.\mathsf{id})$ is the entity represented by t (such mapping is usually implicitly assumed in ER). Then the accuracy of $\mathcal{A}_{\mathsf{ER}}$ on D is traditionally measured in terms of $\mathsf{F}_1 = \frac{2\cdot\mathsf{Precision}\cdot\mathsf{Recall}}{\mathsf{Precision}+\mathsf{Recall}}$. Here Precision is the ratio of *pairs of distinct tuples* that are correctly identified to all identified tuple pairs, *i.e.,* $\mathsf{Precision} = \frac{|\{(t,s)\ |\ t,s\in D,\ \mathcal{A}_{\mathsf{ER}}(t,s)=\mathsf{true},\ f(t.\mathsf{id})=f(s.\mathsf{id})\}|}{|\{(t,s)\ |\ t,s\in D,\ \mathcal{A}_{\mathsf{ER}}(t,s)=\mathsf{true}\}|}$ 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, *i.e.,* $\mathsf{Recall} = \frac{|\{(t,s)\ |\ t,s\in D,\ \mathcal{A}_{\mathsf{ER}}(t,s)=\mathsf{true},\ f(t.\mathsf{id})=f(s.\mathsf{id})\}|}{|\{(t,s)\ |\ t,s\in D,\ f(t.\mathsf{id})=f(s.\mathsf{id})\}|}$.

Example 3: Consider Table 1 of schema $R = (\bar{A}) = (\mathsf{name}, \mathsf{gender}, \mathsf{nationality}, \mathsf{occupation})$, 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 $\mathcal{A}_{\mathsf{ER}}$ makes an FP prediction and an FN prediction as stated in Example 1. The precision of $\mathcal{A}_{\mathsf{ER}}$ on D is $\mathsf{Precision} = \frac{0}{1}$ since the only distinct pair predicted true by $\mathcal{A}_{\mathsf{ER}}$ is (t_1, t_2) (which is a FP). Similarly, $\mathsf{Recall} = \frac{0}{1}$ since the only true match (t_2, t_3) is not identified (due to the FN). □

As shown above, $\mathcal{A}_{\mathsf{ER}}$ 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 $t_G.\mathsf{id} = t.\mathsf{id}$, $t_G[\bar{A}] = t[\bar{A}]$ 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 $\bar{B} = (\mathsf{spouse\_name})$. After enriching R to R_G, with an additional attribute spouse_name, the FN (*i.e.,* (t_2, t_3)) is reduced, as stated in Example 1, improving Precision of $\mathcal{A}_{\mathsf{ER}}$ on D_G to $\frac{1}{2}$, since $\mathcal{A}_{\mathsf{ER}}$ predicts true for both (t_1, t_2) (*i.e.,* the FP) and (t_2, t_3), where only the latter one is correctly identified. Similarly, Recall is also improved to $\frac{1}{1}$ since the only true match (t_2, t_3) in Table 1 is correctly identified. □

In this paper, we use the difference between the F_1 of $\mathcal{A}_{\mathsf{ER}}$ on D_G and on D as *improvement of* $\mathcal{A}_{\mathsf{ER}}$ *on* D *via* D_G. The difference of Precision and Recall can also be used when, *e.g.,* the true negatives (TNs) dominates the solution space in an application.

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

- *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 $\mathcal{A}_{\mathsf{ER}}$ on D via D_G.

Here ER is conducted by the same $\mathcal{A}_{\mathsf{ER}}$ 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 implemented as deleting t followed by inserting a tuple with the changed values. Graph updates, denoted by ΔG, consist of edge insertions/deletions; vertex updates are a dual [71] and can be handled similarly. 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 $\mathcal{A}_{\mathsf{ER}}$ 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 one more attribute born from G of Figure 1, and improve Precision and Recall to 1 (the computation is similar). However, since the information about born of entity e_4 is missing in G, the enriched tuple of t_5 has value null on attribute born. When Table 1 and/or graph G are updated by adding a new edge $e = (v_5, v_{28})$ with $L(e) = \mathsf{born}$ and $L(v_{28}) = 1978$ to G, incremental enrichment dynamically updates the instance D_G of R_G, by updating the born-value of the enriched tuple of t_5 to 1978. □

3.2 Complexity of the Enrichment Problems

We next settle the complexity of the decision problems for enrichment. We show that schema enrichment is NP-complete; hence so is the enrichment problem which subsumes schema enrichment. In contrast, data enrichment and incremental enrichment are tractable, *i.e.,* in polynomial time (PTIME); these two compute D_G and ΔD_G, respectively, after schema R_G is in place. Hence the tricky part is how to select appropriate attributes to enrich schema R.

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

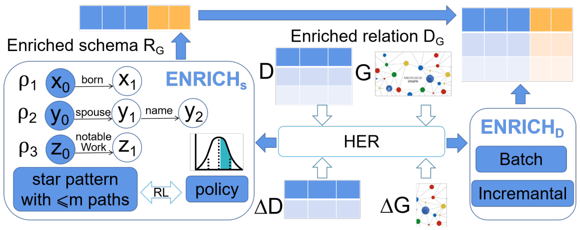


Figure 2: The workflow of ENRICH

exists a set $\bar{B}$ of m attributes such that instance D_G of $R_G = (\bar{A}, \bar{B})$ has accuracy improvement of $\mathcal{A}_{\mathsf{ER}}$ 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, *e.g.,* [24, 41, 77, 86, 89], by reduction from X3C (Exact Cover by 3-sets), which is NP-complete (cf. [52]). 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 $S_i \in C'$. The reductions are nontrivial. In fact we show the NP-hardness also holds for rule-based ER and HER methods [6]. □

3.3 A Scheme for Enrichment

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

The notations of the paper are summarized in Table 2.

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 $\mathcal{A}_{\mathsf{ER}}$, 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 $\mathcal{A}_{\mathsf{ER}}$. 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.

Table 2: Notations

Notations	Definitions
$G = (V, E, L)$	a knowledge graph
$R = (\bar{A})$, $R_G = (\bar{A}, \bar{B})$	a relation schema and its enrichment, respectively
D (resp. D_G)	(resp. enriched) relation of schema R (resp. R_G) with G
ΔD and ΔG	updates to D and G, respectively
ρ_B	a path pattern for representing attribute B
m (resp. k)	the maximum attributes in $\bar{B}$ (resp. edges in ρ_B)
π_θ	a parameterized policy function
$\mathcal{V}_t$	a set of top-K HER matches of tuple t

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* $\mathcal{A}_{\mathsf{ER}}$.

Overview. In light of this, we adopt a policy-learning based approach with a parameterized policy function π_θ (*i.e.,* θ is the set of parameters in π_θ), so that the schema enrichment problem can be solved flexibly in an end-to-end manner, no matter which ER model $\mathcal{A}_{\mathsf{ER}}$ we use. Our approach consists of the following steps.

(1) HER mapping. Taking a training set S 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 first present our HER mapping and policy function in Sections 4.1-4.2, respectively, and then develop SchemaEnr that implements schema enrichment, with effective strategies (Section 4.3).

4.1 Heterogeneous Entity Resolution

We start with a method for the following HER problem.

- *Input:* Schema $R = (\bar{A})$, a training set S of tuples of schema R, a tuple $t \in S$, a knowledge graph G 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 $t \in S$, there are possibly multiple vertices v in G that match t. Our HER method finds top-K most relevant matching vertices. It consists of two steps: (1) *blocking*, which retrieves a set of candidate vertices in G for the given tuple t, and (2) *ranking*, which returns the set $\mathcal{V}_t$ of top-K vertices with the highest correlation strengths, via a ranking strategy.

Blocking. For each tuple $t \in S$, we initialize a set C_t of candidate matches as blocks based on the "similarity" between t and vertices in G; the set C_t consists of vertices whose similarities to t are above a predefined threshold. We compute HER matches within each C_t.

Each block C_t is built via *Jaccard similarity* as follows. We serialize all values of tuple t to a sequence and tokenize it into a set $\text{Set}(t)$. For each vertex $v \in G$, we extract an induced subgraph G_v of G, including v and the neighbors of v. For each vertex 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 $\text{Set}(G_v)$. Then the Jaccard similarity is computed by $\text{Jacc}(t, v) = \frac{|\text{Set}(G_v)) \cap \text{Set}(t)|}{|\text{Set}(G_v)) \cup \text{Set}(t)|}$. Intuitively, t and v may match only if t and G_v share enough keywords.

Ranking. Jaccard similarity only considers word co-occurrence. To find HER matches, within each block, we identify vertices v that match t via parametric simulation [41]. Moreover, for each attribute $A \in \bar{A}$, we find a path pattern ρ_A such that a match of ρ_A pivoted at v in G (if it exists) represents the value $t[A]$.

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 [96], a bert-based model, to transform each vertex v_i in G_v ($i \in [0, l]$) into a high-dimensional 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 $\text{sem}(t, v) = \max_{v_i \in G_v} \cos(\mathbf{e}_t, \mathbf{e}_{v_i})$, where cos is the cosine similarity. We adopt the MLM strategy [33] to pre-train SentBert in training set S and graph G.

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.

<u>Representing $\bar{B}$</u>. 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 $B = L_1 \ldots L_l$, *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. To build $R_G = (\bar{A}, \bar{B})$, we need to find m path patterns ρ_B, so that given a validation set T of schema R, the enriched relation T_G of T computed from path matches of ρ_B ($B \in \bar{B}$) maximumly improve the accuracy of $\mathcal{A}_{\text{ER}}$. There are three criteria for ρ_B.

(1) Diversity. We adopt mutual information $\text{MI}(A, B)$ [22] to measure the correlation between two attributes A and B. Given a validation relation T_G of schema $(\bar{A}, \bar{B})$, we define the diversity of enriched schema on T_G as $\text{div}(T_G) = -\frac{1}{|\bar{A}||\bar{B}|} \sum_{x, y \in \bar{A} \cup \bar{B} \& x \neq y} \text{MI}(x, y)$. Intuitively, we want to enrich the schema 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.,* $\text{comp}(T_G) = -\frac{\#\{\text{null values}\}}{\#\{\text{all values}\}}$. Fewer null values are more desirable.

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

Taken together, the objective value we want to maximize is:
$\text{obj}(T_G, \mathcal{A}_{\text{ER}}) = w_{\text{div}} \text{div}(T_G) + w_{\text{comp}} \text{comp}(T_G) + w_{\text{F}_1} \text{F}_1(T_G, \mathcal{A}_{\text{ER}})$
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 $\text{obj}(T_G, \mathcal{A}_{\text{ER}})$. We approach this problem via *policy learning* so that a robust policy is learned to get the desired set Q.

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 $\rho_{B_1}, \ldots \rho_{B_{i-1}}$ ($i \in [1, m]$), 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 $(\bar{A}, B_1, \ldots, B_i)$. 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.,*

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

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

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

where $\text{transform}(s_{i,j})$ [61] 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 $\mathcal{J}(\theta) = \mathbb{E}_{p(s_{i,j};\theta)}[r_{i,j}]$. Here the reward can be non-differentiable because it is computed based on $\mathcal{A}_{\text{ER}}$ in the validation data. Thus, we use REINFORCE [126], a policy gradient method that imposes no constraints on $\mathcal{A}_{\text{ER}}$, to iteratively update θ of π_θ with

$$\nabla_\theta \mathcal{J}(\theta) = \sum_{x=1}^{i} \sum_{y=1}^{s_{x,\cdot}} \mathbb{E}_{p(s_{x,y};\theta)} [\nabla_\theta \log p(a_{x,y} | s_{x,y-1}; \theta) \cdot r_{x,y}].$$

We adopt an empirical approximation of $\nabla_\theta J(\theta)$ in each batch:

$$\widehat{\nabla_\theta \mathcal{J}_\theta} = \frac{1}{L} \sum_{z=1}^{L} \sum_{x=1}^{i} \sum_{y=1}^{s_{x,\cdot}} \nabla_\theta \log p(a_{x,y} | s_{x,y-1}; \theta) \cdot \text{obj}^M_{x,y},$$

where M is the number of distinct $\bar{B}$ in each batch, L is the batch size, and $\text{obj}^M_{x,y}$ is the objective score after $\mathcal{A}_{\text{ER}}$ is re-trained.

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 $\rho_1 = (x_0, \text{born}, x_1)$, 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 function π_θ is able to construct a set of path patterns without costly path enumeration, it stills encounters the following issues. (1) The distributions of the training and validation sets keep changing due to schema enrichment, and it is costly to frequently re-train $\mathcal{A}_{\text{ER}}$. Worse still, (2) the efficiency of policy learning depends on the feedback from the ER model $\mathcal{A}_{\text{ER}}$; this would make 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 $\mathcal{A}_{\text{ER}}$, making up the time for computing feedbacks from $\mathcal{A}_{\text{ER}}$ 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 $\mathcal{A}_{\text{ER}}$, a maximum inference number δ, a maximum batch number I, three parameters m, k and K for constraining the maximum number of additional attributes, the length of path patterns and the number of HER matches, respectively, we provide SchemaEnr in Figure 3. It returns an enriched schema $R_G = (\bar{A}, \bar{B})$ such that the objective value is maximized on the enriched validation data. Here δ is a configurable parameter and it controls the number of candidate path pattern sets that we can generate in the inference step.

After initializing the parameters of π_θ (line 1), SchemaEnr precomputes the top-K HER matches in G for each tuple in S (resp. T, lines 2-3). Following [83], we *jointly* optimize π_θ and $\mathcal{A}_{\text{ER}}$ in *batches* (line 4-17) such that the policy function learns to find "good" path patterns and the ER model is fine-tuned to improve accuracy on the enriched relations computed based on the HER matches.

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 $\mathcal{A}_{\text{ER}}$ (lines 7-12). In the i-th iteration, a new path $\rho_{B_i}^{\text{bat}}$ 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 $|\rho_{B_i}^{\text{bat}}| > k$. A new attribute B_i^{bat} is created accordingly by concatenating the edge labels of $\rho_{B_i}^{\text{bat}}$ (line 8). Note that even when one more attribute is added, the distribution of enriched training/validation data may change dramatically. To make $\mathcal{A}_{\text{ER}}$ robust to diverse distributions, we accumulate the enriched training (resp. validation) data in a set $\mathcal{S}_{\text{train}}$ (resp. $\mathcal{T}_{\text{valid}}$), initially empty (line 5), during the iterative process (lines 9-11). Whenever we get a new attribute B_i^{bat}, we compute the enriched relations of $S_{\text{train}}/T_{\text{valid}}$, where the B_i^{bat}-value of the enriched tuple of t is computed by following $\rho_{B_i}^{\text{bat}}$, starting form a HER match in $\mathcal{V}_t$ (see Section 5.1). The enriched sets of S_{train} and T_{valid} are then added to $\mathcal{S}_{\text{train}}$ and $\mathcal{T}_{\text{valid}}$, respectively. Finally, the entire $\mathcal{S}_{\text{train}}$ is adopted to upgrade $\mathcal{A}_{\text{ER}}$ with the cross entropy loss (line 12).

Input: A schema $R = (\bar{A})$, a training set S, a validation set T, a graph G, $\mathcal{A}_{\text{ER}}$, an inference number δ, 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 π_θ; bat := 0;
2. **for each** t in S or T **do**
3. $\quad\mathcal{V}_t$:= the top-K HER matches of t in G;
4. **while** $\widehat{\nabla_\theta \mathcal{J}_\theta}$ does not converge and bat $< I$ **do**
5. $\quad S_{\text{train}}$:= getBatch(S); T_{valid} := getBatch(T); $\mathcal{S}_{\text{train}} := \mathcal{T}_{\text{valid}} := \emptyset$;
6. $\quad\bar{B}^{\text{bat}} := \emptyset$; /* The enriched schema for the current batch */
   /*Joint training: Fix policy π_θ and train $\mathcal{A}_{\text{ER}}$ */
7. $\quad$**for each** $i \in [1, m]$ **do**
8. $\quad\quad\rho_{B_i}^{\text{bat}}$:= 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. $\quad\quad\Delta_{\text{train}}$:= the enriched relation of S_{train} under schema $(\bar{A}, \bar{B}^{\text{bat}})$;
10. $\quad\quad\Delta_{\text{valid}}$:= the enriched relation of T_{valid} under schema $(\bar{A}, \bar{B}^{\text{bat}})$;
11. $\quad\quad\mathcal{S}_{\text{train}} := \mathcal{S}_{\text{train}} \cup \Delta_{\text{train}}$; $\mathcal{T}_{\text{valid}} := \mathcal{T}_{\text{valid}} \cup \Delta_{\text{valid}}$;
12. $\quad$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 π_θ */
13. $\quad Q$:= SampleQ(π_θ) where Q has m paths $\rho_{B_1}, \ldots, \rho_{B_m}$;
14. $\quad$**for each** state $s_{i,j} = Q_{i,j}$ when generating Q with π_θ **do**
15. $\quad\quad$rw_sum := $\sum_{s=|Q_{i,j}|}^{|Q|} \gamma^{I-|Q_{i,j}|} \cdot r_s$, where γ is the decay factor and r_s is the reward at state s, *i.e.,* r_s = Reward(s);
16. $\quad\quad\theta := \theta + \alpha \cdot$ rw_sum $\cdot \nabla_\theta \log \pi_\theta(a_{i,j} | s_{i,j})$ with learning rate α;
17. $\quad$bat := bat + 1;
18. $\bar{B}$:= Inference$(\bar{A}, \pi_\theta, \delta)$;
19. **return** $R_G = (\bar{A}, \bar{B})$;

Figure 3: Algorithm SchemaEnr

Then we fix $\mathcal{A}_{\text{ER}}$ 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 rewards computed via procedure Reward (Line 14-16, 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 $\mathcal{A}_{\text{ER}}$ are optimized iteratively until $\widehat{\nabla_\theta \mathcal{J}_\theta}$ converges or it reaches the maximum number I of batches. Eventually, we obtain the set $\bar{B}$ of additional attributes, by calling procedure Inference (lines 18, see below). With a relatively small learning rate α and consistently convergent $\mathcal{A}_{\text{ER}}$, π_θ will eventually converge, and at least to a local minima [91, 114], *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 whose completeness is small, *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 $\mathcal{A}_{\text{ER}}$ is not stable in the first few epochs, we design a *warm-up* strategy, such that a small weight w_{F1} is set 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 $\mathcal{A}_{\text{ER}}$. Then w_{F1} (resp. w_{div} and

w_{comp}) gradually increases (resp. decrease) until they become 1.

Procedure Inference. Given $\bar{A}$, the learned policy π_θ and the maximum inference number δ, Inference computes the final set $\bar{B}$ of at most m attributes, by making δ rounds of inference; in the i-th round of inference, we generate a candidate set Q_i of pattern paths, following the policy π_θ. More specifically, in the first round, the set Q_1 is obtained by performing the actions with the maximum rewards, while in the remaining rounds, the set Q_i ($i > 1$) is constructed by sampling from the action probability of π_θ. Given the δ sets of candidate path pattern sets, the final set $\bar{B}$ is obtained from the one with the maximum objective value on the validation data.

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, $\mathcal{A}_{\text{ER}}$ is first trained on S. Due to the lack of initial attributes, $\mathcal{A}_{\text{ER}}$ does not work well. Then SchemaEnr executes SampleQ to sample a few path patterns, *e.g.,* $\rho_1 = (x_0, \text{born}, x_1)$, $\rho_2 = (y_0, \text{spouse}, y_1, \text{name}, y_2)$, $\rho_3 = (z_0, \text{notableWork}, y_1)$ and $\rho_4 = (u_0, \text{belongsTo}, u_1, \text{notableWork}, u_2)$. 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_3. 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 $\mathcal{A}_{\text{ER}}$ 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 $O((|S| + |T|)|G|\text{Epoch}_{\max} mk)$ time, when it takes $\text{Epoch}_{\max}$ epochs to train π_θ and $\mathcal{A}_{\text{ER}}$. The HER mapping takes at most $O((|S|+|T|)|G|)$ 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 of policy learning by making up the time for fine-tuning $\mathcal{A}_{\text{ER}}$ in each epoch, *e.g.,* it takes 2,213s to learn the policy on a dataset with 3,162 tuples in 10 epoches.

5 POPULATING ENRICHED SCHEMA

In this section, we develop algorithms for populating and maintaining relations D_G of R_G after the enriched schema $R_G = (\bar{A}, \bar{B})$ is computed (along with the path pattern ρ_B for each B in $\bar{B}$), by referencing a reliable knowledge graph G. We develop a batch algorithm BEnrich (Section 5.1) and an incremental IncEnrich (Section 5.2). They are parallelized as PBEnrich and PIncEnrich, respectively [6].

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 in G for each tuple t in D; and (2) *Populating*, which instantiates the $\bar{B}$-attribute values of tuples in D with G to get the enriched relation D_G. Step (1) has been presented in Section 4.1 and we focus on step (2) below.

Populating. For each tuple t in D, we create an enriched tuple t_G as follows. (a) For each $A \in \bar{A}$, $t_G[A]$ copies $t[A]$; and (b) for each $B \in \bar{B}$, we compute a set $C_{t_G[B]}$ of candidate values for $t_G[B]$:

- If $C_{t_G[B]}$ is an empty set, we set $t_G[B]$ = null.
- If $C_{t_G[B]}$ is not empty, we adopt a ranking model to retrieve the top-ranked value from $C_{t_G[B]}$ and assign it to $t_G[B]$.

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 in the attribute. On the one hand, a longer ρ_B may lead to more combinations of edge labels and thus, more candidate attributes B to be selected. 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 be instantiated with null values, if we cannot find such path matches. To strike a good balance, we use the parameter k to bound the length of paths. We will test the impact of k in Section 6.

A ranking model. We train a model $\mathcal{M}_{\text{rank}}(t_G, \bar{A}, B)$ to rank the candidate values of $C_{t_G[B]}$ for the B-attribute of t_G. Following [48], we use an encoder ENC_{attr} (resp. $\text{ENC}_{\text{value}}$) that transforms the B-attribute (resp. each value c in $C_{t_G[B]}$) to a high-dimensional embedding ϕ_B (resp. ϕ_c). ENC_{attr} and ENC_{val} are Bert-based models that share the same parameters but adopt different fully-connected layers. We then rank the embeddings ϕ_c by their "distances" to ϕ_B, *i.e.,* we use the embedding ϕ_B as the target and make the embedding of a more suitable value for $t_G[B]$ closer to the target.

To train ENC_{attr} and ENC_{val}, we use an automatic labeling strategy as follows. Consider the B-attribute when enriching tuple t. We randomly select a subset S_t of tuples from the training set S of schema R. Then we adopt mutual information (MI) as a surrogate function to compute the gain of a candidate value c towards the ER label for each tuple pair (t_a, t), where t_a is a tuple in S_t [27]; here the ER label is 1 if (t_a, t) refers to the same entity and is 0 otherwise. More specifically, we enrich t with $t[B] = c$. Then we compute the mutual information of value c in attribute B and the ER labels of all pairs (t_a, t) ($t_a \in S_t$), denoted by $\text{MI}(c, t, S_t)$. Given two candidates $c_1, c_2 \in C_{t_G[B]}$, if $\text{MI}(c_1, t, S_t)) > \text{MI}(c_2, t, S_t))$, c_1 is ranked above c_2, and vice versa. We adopt the triplet loss [48] to fine-tune ENC_{attr} and ENC_{val}. We let $t[B]$ be the top-ranked c.

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.,* $h_1 : \{(v_3, v_{15}) \mapsto (x_0, x_1)\}$ is the only path match of ρ_1 pivoted at v_3 and thus, the born-value of the enriched tuple of t_3 is 1971 by the ranking model. In contrast, ρ_3 finds no path match pivoted at v_3 and thus, the notableWork-value of the enriched tuple of t_3 is null. □

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.

Remark. Although populating with HER mapping can also be regarded as a matching task, it is often more accurate than ER, since it not only consider top-K candidate HER matches, but also adopts a policy function and ranking model whose objectives are to improve the accuracy of $\mathcal{A}_{\text{ER}}$ to decide what attributes should be enriched and what values should be assigned, respectively.

5.2 Incremental Enrichment

We next develop the incremental algorithm IncEnrich. The need for IncEnrich is evident. Real-life datasets and knowledge graphs constantly change. For example, Wikidata [5] publishes hundreds of live updates every minute [4]. It is too costly to populate enriched relations starting from scratch in response to the updates.

Setting. We consider both graph updates ΔG and relation updates ΔD, where ΔD consists of deleted/inserted tuples and ΔG consists of edge. 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 handled by the batch algorithm. Below we focus on part (b), which computes the updates on the enriched relation D_G when G is updated by ΔG.

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 to 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).

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.

<u>*Unit insertion.*</u> When an edge e is inserted into G, we create an new entry, denoted by $\mathsf{Piv}(e)$, 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 $\mathsf{Piv}(e)$. 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 matching in BEnrich. For each tuple t whose top-K HER matches includes v_0, we update the set $C_{t_G[B]}$ of candidate values, by adding the last vertex label of $h(\rho_B)$ and call the ranking model $\mathcal{M}_{\mathsf{rank}}$ to get the new top-ranked value for the B-attribute of 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

Input: An enriched relation D_G, a knowledge graph G, schema $R_G = (\bar{A}, \bar{B})$, graph updates ΔG and the auxiliary structures.
Output: The updates of the enriched relation of D with $G \oplus \Delta G$.

1. $P := \mathsf{GetAffectedPathMatches}(\Delta G, G, \mathsf{Piv})$;
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 ρ_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 ρ_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** $\{t_G \in D_G \mid t_G \text{ is updated}\}$;

Figure 4: Algorithm IncEnrich

ΔG that inserts a new edge $e = (v_5, v_{28})$ into G, where $L(e) = \mathsf{born}$ and $L(v_{28}) = 1978$. Given the path pattern $\rho_1 = (x_0, \mathsf{born}, x_1)$ in Figure 2, $h : \{(v_5, v_{28}) \mapsto (x_0, x_1)\}$ is a path match of ρ_1. Thus, we add the pivot v_5 to $\mathsf{Piv}(e)$ and compute $P_{v_5} = \{h(\rho_1)\}$. Since $h(\rho_1)$ is a path match of Case **[C1]**, v_5 is still an HER match of t_5 in D and we can populate the born-value of the enriched tuple of t_5 by $L(v_{28})$, *i.e.,*1978. For ΔD, we simply run $\mathsf{BEnrich}(\Delta D, G \oplus \Delta G)$ to populate the $\bar{B}$-attributes of the enriched tuple of t_6. □

<u>*Unit deletion.*</u> 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 $h \in P_{v_0}$ as follows. (1) **[C1] h is a path match of ρ_B.** For each tuple t whose top-K HER matches includes v_0, we update $C_{t_G[B]}$ by removing the value introduced by $h(\rho_B)$, and update the assignment of $t_G[B]$ based on the ranking model accordingly. In particular, if $C_{t_G[B]}$ becomes empty, we simply set $t_G[B] = \mathsf{null}$. (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.

<u>*Batch updates.*</u> Based on unit updates, we develop IncEnrich in Figure 4, 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 $\mathsf{Piv}[e]$ for all $e \in \Delta G$ (line 1). Then it groups the affected path matches by their pivots [42], 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 tuple t whose top-K HER matches include v_0, and update the B-attribute 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 it if v_0 is no longer one of the top-K HER matches (resp. a candidate match) of t. If $\mathcal{V}_t$ is changed, we re-populate all $\bar{B}$-attribute values of t_G based on a new set of candidate values from $\mathcal{V}_t$. Finally, the updates of the enriched relation of D with $G \oplus \Delta G$ are returned (line 12).

<u>*Complexity.*</u> IncEnrich takes $O(c_{\mathsf{up}}\#\mathsf{Aff}|\Delta G||P_e|)$ 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 $O(c_{\mathsf{up}}\#\mathsf{Aff})$ 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 real-life and synthetic data, we empirically evaluated (1) the effectiveness of schema enrichment (SE) and the impact of our policy function on the 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 two benchmarks and two real-life datasets D. Table 3 reports the number of tuples, the size of $\bar{A}$, and the KG G for each dataset. Here (1) Shoes [77, 86] is an ER benchmark from WDC Product. (2) Amazon [77] is a benchmark of product data. For the two ER benchmarks, we used all their attributes as $\bar{A}$, and follow the same setting of training data S, validation data T and testing data U of [77]. (3) Person [3] is a real-life dataset of person tuples from Wikipedia. (4) IMDB [1] is a real-life dataset that contains movies and TV Series from 1905 to 2022. We adopted the Jaccard similarity to retrieve candidate pairs and manually labeled matches.

ER model $\mathcal{A}_{\text{ER}}$. We used three deep learning $\mathcal{A}_{\text{ER}}$ models: (1) Ditto [77], a state-of-the-art pre-trained language model. We used RoBerta [80] for Ditto without data augmentation. (2) $\text{Ditto}_{\text{aug}}$ [77], Ditto with data augmentation. (3) PromptEM [113], a state-of-the-art ER model that adopts prompt tuning to fine-tune pre-trained language model. We adopted RoBerta [80] following [113].

Hyper-parameters. We adopted the CNN architecture with a fully connected layer of 128 dimension for π_θ. The learning rate is 3e-4. The batch size of the training set S (resp. the validation set T) 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 paths in graph G, $K = 3$ for the number of HER matches in $\mathcal{V}_t$, $I = 200$ (resp. $\delta = 10$) for the batch (resp. inference) number. For HER, we sampled 30K, 30K, 30K and 50K tuples from D and paths from G to pre-train SentBert in Shoes, Amazon, Person and IMDB, respectively.

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 [77], we truncated the enriched features to the maximum size. (4) MI [21], a heuristic method that greedily selects m paths as the enriched attributes to maximize the mutual information from G. (5) AutoFeature [79], 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 [27], 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 tuple 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 fair comparisons, we use the same HER algorithm (Section 4.1) and the inference strategy (Section 4.3) for all baselines whenever possible.

Table 3: The datasets and knowledge graphs

Datasets	$\lvert D\rvert$	$\lvert\bar{A}\rvert$	G	$\lvert V\rvert$	$\lvert E\rvert$
Shoes [77]	3162	3	Wikidata [2]	1.1M	6.3M
Amazon [77]	4589	3	Wikidata [2]	1.1M	6.3M
Person [3]	2.7M	3	Wikidata [2]	1.1M	6.3M
IMDB [1]	2.0M	3	Movie [1]	6.1M	30.0M

Measures. (1) For SE, we report the running time and the accuracy of each SE method in the testing set. After SE, we generated $\bar{B}$ for each dataset. (2) For BE and IE, we report the runtime of each method which populates the $\bar{B}$-attribute values by referencing G.

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.

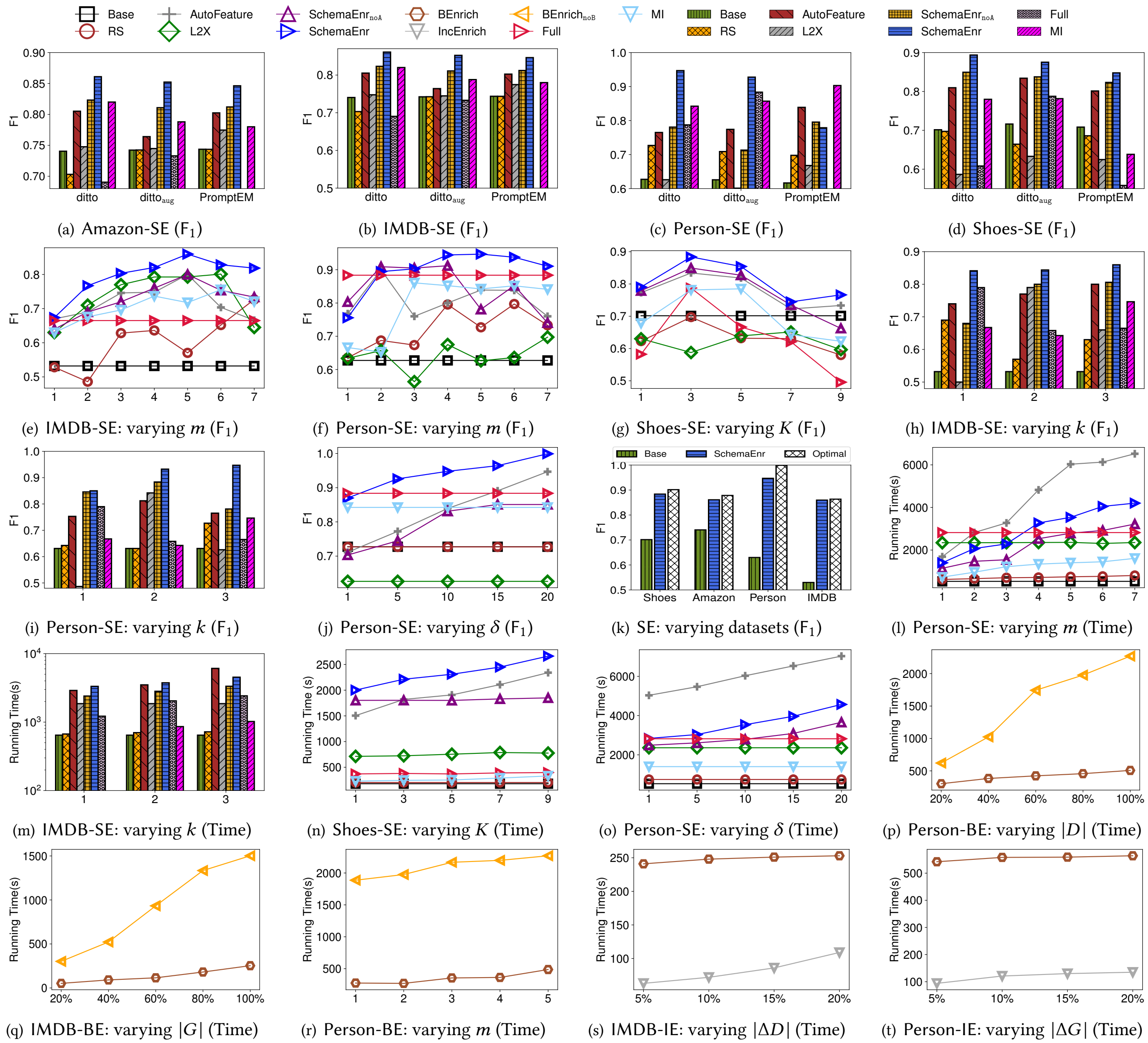
Experimental results. We next report our findings.

Exp-1 Effectiveness. We evaluated SchemaEnr in terms of (1) the accuracy F_1 in the testing set after $R = (\bar{A})$ is enriched with $\bar{B}$; (2) the impact of increasing attributes and lengths of paths for $\mathcal{A}_{\text{ER}}$; and (3) the gap between $\bar{B}$ enriched in SchemaEnr and the optimal $\bar{B}_{\text{opt}}$. Note that it is infeasible to compute $\bar{B}_{\text{opt}}$ when m is large. Thus, we first manually selected 30 path patterns and filtered vertices leading to low rewards, and then enumerated the set P_{all} of all paths in G from the remaining pivots that are mapped to some tuples in D. Then we iteratively enriched training/validation sets with at most m attributes constructed from the paths in P_{all}, trained $\mathcal{A}_{\text{ER}}$ and got the F_1 of $\mathcal{A}_{\text{ER}}$. The optimal $\bar{B}_{\text{opt}}$ is the one with the largest F_1.

Accuracy vs. baselines. We tested SchemaEnr in Figure 5(a)-5(d).

(1) SchemaEnr is 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 20.6%, 25% and 17.2% 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. (b) The learned policy function π_θ is able to find high-quality paths from G ($\bar{B}$-attributes) better 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 $\mathcal{A}_{\text{ER}}$. (d) RS does not always outperform Base, *e.g.,* the F_1 of RS (resp. Base) is 0.49 (resp. 0.53) on IMDB when $m = 2$. As discussed in Section 4, some vertices in G mapped by tuples in D do not have certain specific paths and hence, some $\bar{B}$ attributes of the tuples have null values, which deteriorate the performance of $\mathcal{A}_{\text{ER}}$.



(a) Amazon-SE (F_1) (b) IMDB-SE (F_1) (c) Person-SE (F_1) (d) Shoes-SE (F_1)

(e) IMDB-SE: varying m (F_1) (f) Person-SE: varying m (F_1) (g) Shoes-SE: varying K (F_1) (h) IMDB-SE: varying k (F_1)

(i) Person-SE: varying k (F_1) (j) Person-SE: varying δ (F_1) (k) SE: varying datasets (F_1) (l) Person-SE: varying m (Time)

(m) IMDB-SE: varying k (Time) (n) Shoes-SE: varying K (Time) (o) Person-SE: varying δ (Time) (p) Person-BE: varying $|D|$ (Time)

(q) IMDB-BE: varying $|G|$ (Time) (r) Person-BE: varying m (Time) (s) IMDB-IE: varying $|\Delta D|$ (Time) (t) Person-IE: varying $|\Delta G|$ (Time)

Figure 5: Performance evaluation

(3) SchemaEnr consistently outperforms MI, AutoFeature and L2X, *e.g.*, its F_1 is 9%, 7% and 18% higher than the baselines on average for the three, respectively. This is because SchemaEnr finds high-quality paths to improve the ER model $\mathcal{A}_{ER}$ with the warm-up and mask strategies, while AutoFeature is not designed for path selection in G, and MI selects features that are independent from $\mathcal{A}_{ER}$. Although L2X selects important features for learning $\mathcal{A}_{ER}$, the premise of L2X is that features, *i.e.*, all paths in G, should be enumerated and prepared before features are selected. However, enumerating all paths in IMDB is infeasible and L2X does not work well in most cases. In contrast, SchemaEnr conducts reinforcement learning without enumerating all paths. Moreover, SchemaEnr is able to support any $\mathcal{A}_{ER}$ while L2X requires $\mathcal{A}_{ER}$ to be differentiable.

Varying m. We varied $m = |\bar{B}|$ from 1 to 7 in Figures 5(e)-5(f). 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 $\mathcal{A}_{ER}$ by adding distinguishing attributes from G. However, its F_1 drops as m continues to increase, *e.g.*, it reduces to 0.819 when $m = 7$. This is because when m reaches, *e.g.*, 5 on IMDB, the contextual information is enough to learn $\mathcal{A}_{ER}$ 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. The F_1 of the baselines may fluctuate when m is large, *e.g.*, L2X changes from 0.6337 to 0.675 and then to 0.626 when m is 1, 4 and 5 on Person. This is because most baselines aim to maximize simple policies, *e.g.*, the mutual information of L2X, and their features might damage $\mathcal{A}_{ER}$.

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

Varying k. As shown in Figure 5(h)-5(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 in Person. Although the ratio of null values slightly increases as k increases, *e.g.,* 35%, 38% and 39% for $k = 1$ (*i.e.,* $\text{SchemaEnr}_{k=1}$), 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 of the baselines, fails to find 3-hop paths in IMDB and Person because it cannot extract fine-grained paths in graphs.

Varying δ. As shown in Figure 5(j), we varied the inference number δ from 1 to 20. The F_1 of SchemaEnr increases as δ increases, *e.g.,* from 0.8699 to 0.9991 when δ is changed from 1 to 20. This is because SchemaEnr is able to explore more candidate path pattern sets, so as to pick the best one among them in the inference step.

Varying datasets. In Figure 5(k), we varied the datasets and checked whether $\bar{B}$ generated by SchemaEnr is as effective as $\bar{B}_{\text{opt}}$. Although $\bar{B}$ and $\bar{B}_{\text{opt}}$ are not exactly the same over some datasets, *e.g.,* their Jaccard similarity $\text{Jacc}(\bar{B}, \bar{B}_{\text{opt}})$ is 67% in IMDB, their accuracy does not differ much, *i.e.,* the gap between F_1 is as small as 0.004. This verifies SchemaEnr is effective enough to learn the policy that finds distinguishing attributes from G. The gaps of Person, Shoes and Amazon are not as close as IMDB, because the KGs they used are much denser than that of IMDB and thus, it is more difficult to find $\bar{B}$ in these datasets. This said, the gaps are at most 5% from $\bar{B}_{\text{opt}}$.

Exp-2 Efficiency. We evaluated the running time (incl. both the training time and the inference time) of SchemaEnr.

Varying m. As shown in Figure 5(l) 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 in Person. This justifies the need for setting budget m for enrichment. SchemaEnr is not the fastest learner, *e.g.,* its running time 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, since these baselines are simply based on heuristic policies with the price of lower accuracy.

Varying k. In Figures 5(m), 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 [11, 66] that longer paths hold weaker associations. RS, Full and MI are fast because they use simple policies.

Varying K. We varied K from 1 to 9 in Figure 5(n). 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.

Varying δ. In Figure 5(o), we varied δ from 1 to 20. The running time of SchemaEnr increases slightly as δ increases, *e.g.,* the runtime of $\delta = 1$ is 1.25X faster than $\delta = 10$. Although SchemaEnr explores more candidate path pattern sets in the inference step, the training time still dominates the whole process. Thus we pick a value for δ, *e.g.,* 10, to strike a balance between the cost and accuracy.

Joint training. We also revised SchemaEnr by iteratively training π_θ and $\mathcal{A}_{\text{ER}}$ separately, and compared it with 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. $\text{BEnrich}_{\text{noB}}$ 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 $\text{BEnrich}_{\text{noB}}$ in Figure 5(p). Both take longer with larger D because they need to enrich more tuples from knowledge graphs. Nonetheless, BEnrich is 3.68X faster than $\text{BEnrich}_{\text{noB}}$ on average, which verifies the need for efficient HER methods.

Varying $|G|$. As shown in Figure 5(q) by varying $|G|$ from 20% to 100%, the runtime of all methods increases when $|G|$ gets larger, *e.g.,* BEnrich takes 52s and 181s when $|G|$ is 20% and 80%, respectively. BEnrich is still 6.4X faster than the baseline on average.

Varying m. Varying m from 1 to 5 in Figure 5(r), 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 5(s). IncEnrich constantly beats its batch counterpart BEnrich. On average IncEnrich is 3.1X faster than BEnrich when $|\Delta D|$ varies from 5% to 20%; it is 3.8X faster when $|\Delta D| = 5\%|D|$. The results are expected because IncEnrich enrich only tuples in ΔD, not the entire D. Note that it is more costly to handle ΔG (= 10%) than ΔD, because $|G|$ is much larger than $|D|$ in IMDB and computing affected paths is costly.

Varying $|\Delta G|$. Fixing $|\Delta D|$ = 10%, we varied the number of edge updates $|\Delta G|$ to G in Figure 5(t). IncEnrich beats BEnrich by 4.71X on average when ΔG varies from 5% to 20%, and by 5.77X 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 in IMDB with 4 more attributes. (2) It consistently outperforms the baselines, *e.g.,* on average it is 9%, 7% and 18% more accurate than MI, AutoFeature and L2X, respectively. (3) It beats all its variants, verifying *e.g.,* the benefit of joint training vs. $\text{SchemaEnr}_{\text{noA}}$. (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 in IMDB when $K = 3$ and it is only 1.77X 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 5.77X faster than BEnrich.

7 RELATED WORK

We categorize the related work as follows.

Feature augmentation. Prior work on the topic is classified as follows. (1) *Join-based methods.* [35, 73, 98, 100, 125] enrich tables by joining external tables in data lakes. (2) *Table discovery and union search methods.* PEXSEO [35] proposes a framework for joinable table discovery via similarity join. Josie [125] designs an overlap set similarity method to find joinable tables. [122] discovers related tables in a human-in-the-loop manner. COCOA [37] adopts a lightweight index to accelerate tabular enrichment based on non-linear correlation measures. [67] and [88] find tables that are unionable in data lakes based on the semantics of metadata or correlations between attributes in tables. (3) *Knowledge based methods*, to enrich tabular data with knowledge graphs [49, 55, 85], unstructured textual data [55, 56], information space [34], data warehouse [12], structured Web data [51] and rule injection [77]. (4) *ML based methods.* AugDiff [101] proposes a diffusion-based feature augmentation framework for multiple instance learning. [28] adopts the auto-encoder neural network to transform raw images to semantic vectors with augmented features. [123] proposes spectral feature augmentation to boost contrastive learning. [76] adopts transformation functions for augmentation, by adding random variables from predefined distributions. (5) *Model-aware methods*, for optimizing downstream models, *e.g.,* AutoFeature [79] applies multi-armed bandit and DQN strategies to get useful features for ML models, [112] designs coreset selection methods to make ML feature rich without materializing augmented tables, and [73, 100] explore key-foreign key joins on ML classifiers and adopt feature selection techniques to predict when joins can be avoided safely.

This work differs from the prior work in the following. (1) While some existing methods also incorporate external knowledge (*e.g.,* knowledge graphs [49, 55, 85]) for feature augmentation, they are not application-aware, while we enrich incomplete schema with bounded attributes to maximize the accuracy of ER. Moreover, our method can be easily adapted to support other types of external knowledge, not just knowledge graphs. (2) Although [73, 79, 112] optimize for downstream models, they target at routine models, not black-box ER models, and focus on finding coarse-grained joinable tables in data lakes, whose schemas are already in place. In contrast, we extract additional fine-grained attributes via paths from knowledge graphs 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 [54], mutual information [21, 27, 69], relief [111], markov blanket [63, 128], 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 [10, 104] and evolutionary algorithms [84, 107, 109]. 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 [108], Ridge [58] and Elastic Net [127]. 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 the 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 also proposed to incorporate various knowledge. (1) *Internal knowledge,* to impute missing values using data dependencies, *e.g.,* FDs, CFDs [40], DCs [16], PFDs [92] and REEs [46, 47]. There are also ML-based imputation methods, *e.g.,* Baran [81], HoloClean [97, 117], PClean [74] and Restore [57] for relational tables, ORBITS [68] and DeepMVI [20] for time series, and GAIN [118] and GINN [106] for images. (2) *Master data.* [31, 39, 44] correct errors in relations by referencing master data with correctness guarantees. [46] adopts the chase to correct errors, with the Church-Rosser property. (3) *Knowledge graph.* FROG [93] proposes imputation methods with complex semantics and designs an index to accelerate value retrieval from knowledge graphs. HER methods, *e.g.,* JedAI [90], parametric simulation [41], Silk [62] and MAGNN [50], conduct heterogeneous entity resolution to link tuples in relational tables to vertices in graph. Other entity linking methods, *e.g.,* [78, 95], link mentions of texts to vertices in graphs (see [99] for a survey). (4) *Large language models.* [87] uses GPT-3 to wrangle relational tables, *i.e.,* imputing missing values in a generative manner 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 deep-learning based ER models (see a survey in [29]), which adopt neural networks, attention, RNN and pretrained language models to identify whether two tuples refer to the same entity, *e.g.,* DeepMatcher [86], DeepER [36] Ditto [77], AutoEM [124], BertER [75], DADER [110], PromptEM [113]. These models can be plugged in our scheme as downstream ER methods.

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.

REFERENCES

[1] 2019. IMDB. *https://www.imdb.com/interfaces/*.
[2] 2022. DBpedia. *http://wiki.dbpedia.org*.
[3] 2022. Wikemedia. https://www.kaggle.com/datasets/kenshoresearch/kensho-derived-wikimedia-data.
[4] 2022. Wikidata – Recent changes. *https://www.amazon.science/blog/combining-knowledge-graphs-quickly-and-accurately*.
[5] 2022. Wikipedia. *https://www.wikipedia.org*.
[6] 2023. Code, datasets and full version. https://www.dropbox.com/sh/kk84zjrgwa0dikc/AABZ6MueJGc03MsZIbw8k-cra?dl=0.
[7] 2023. Social Network Usage and Growth Statistics. *https://backlinko.com/social-media-users*.
[8] 2023. Wikidata:WikiProject Disambiguation pages. https://www.wikidata.org/wiki/Wikidata:WikiProject_Disambiguation_pages.
[9] Ghadeer Abuoda, Saravanan Thirumuruganathan, and Ashraf Aboulnaga. 2022. Accelerating Entity Lookups in Knowledge Graphs Through Embeddings. In *ICDE*. IEEE, 1111–1123.
[10] 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.
[11] Boanerges Aleman-Meza, Christian Halaschek-Wiener, Ismailcem Budak Arpinar, and Amit P. Sheth. 2003. Context-Aware Semantic Association Ranking. In *SWDB*. 33–50.
[12] Rohit Ananthakrishna, Surajit Chaudhuri, and Venkatesh Ganti. 2002. Eliminating Fuzzy Duplicates in Data Warehouses. In *VLDB*. 586–597.
[13] Renzo Angles, Angela Bonifati, Stefania Dumbrava, George Fletcher, Keith W. Hare, Jan Hidders, Victor E. Lee, Bei Li, Leonid Libkin, Wim Martens, Filip Murlak, Josh Perryman, Ognjen Savkovic, Michael Schmidt, Juan F. Sequeda, Slawek Staworko, and Dominik Tomaszuk. 2021. PG-Keys: Keys for Property Graphs. In *SIGMOD*. 2423–2436.
[14] Arvind Arasu, Michaela Götz, and Raghav Kaushik. 2010. On active learning of record matching packages. In *SIGMOD*. 783–794.
[15] Arvind Arasu, Christopher Ré, and Dan Suciu. 2009. Large-Scale Deduplication with Constraints Using Dedupalog. In *ICDE*. 952–963.
[16] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. 1999. Consistent Query Answers in Inconsistent Databases. In *PODS*. 68–79.
[17] Abolfazl Asudeh, Nima Shahbazi, Zhongjun Jin, and H. V. Jagadish. 2021. Identifying Insufficient Data Coverage for Ordinal Continuous-Valued Attributes. In *SIGMOD*. 129–141.
[18] Zeinab Bahmani and Leopoldo E. Bertossi. 2017. Enforcing Relational Matching Dependencies with Datalog for Entity Resolution. In *FLAIRS*.
[19] 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.
[20] Parikshit Bansal, Prathamesh Deshpande, and Sunita Sarawagi. 2021. Missing Value Imputation on Multidimensional Time Series. *PVLDB* 14, 11 (2021), 2533–2545.
[21] Roberto Battiti. 1994. Using mutual information for selecting features in supervised neural net learning. *IEEE Trans. Neural Networks* 5, 4 (1994), 537–550.
[22] 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.
[23] 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.
[24] Indrajit Bhattacharya and Lise Getoor. 2006. Entity resolution in graphs. *Mining graph data* 311 (2006).
[25] Indrajit Bhattacharya and Lise Getoor. 2007. Collective entity resolution in relational data. *TKDD* 1, 1 (2007), 5.
[26] 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.
[27] 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.
[28] 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.
[29] 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.
[30] E. F. Codd. 1979. Extending the Database Relational Model to Capture More Meaning. *ACM Trans. Database Syst.* 4, 4 (1979), 397–434.
[31] 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.
[32] 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.
[33] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*. 4171–4186.
[34] Xin Dong, Alon Y. Halevy, and Jayant Madhavan. 2005. Reference Reconciliation in Complex Information Spaces. In *SIGMOD*. ACM, 85–96.
[35] 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.
[36] 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.
[37] Mahdi Esmailoghli, Jorge-Arnulfo Quiané-Ruiz, and Ziawasch Abedjan. 2021. COCOA: COrrelation COefficient-Aware Data Augmentation.. In *EDBT*. 331–336.
[38] Wenfei Fan, Hong Gao, Xibei Jia, Jianzhong Li, and Shuai Ma. 2011. Dynamic constraints for record matching. *VLDB J.* 20, 4 (2011), 495–520.
[39] Wenfei Fan and Floris Geerts. 2010. Relative information completeness. *ACM Trans. Database Syst.* 35, 4 (2010), 27:1–27:44.
[40] 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.
[41] 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.
[42] Wenfei Fan, Chunming Hu, and Chao Tian. 2017. Incremental Graph Computations: Doable and Undoable. In *SIGMOD*. 155–169.
[43] Wenfei Fan, Xibei Jia, Jianzhong Li, and Shuai Ma. 2009. Reasoning about Record Matching Rules. *PVLDB* 2, 1 (2009), 407–418.
[44] 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.
[45] Wenfei Fan and Ping Lu. 2019. Dependencies for Graphs. *ACM Trans. Database Syst.* 44, 2 (2019), 5:1–5:40.
[46] Wenfei Fan, Ping Lu, and Chao Tian. 2020. Unifying Logic Rules and Machine Learning for Entity Enhancing. *Sci. China Inf. Sci.* 63, 7 (2020).
[47] Wenfei Fan, Chao Tian, Yanghao Wang, and Qiang Yin. 2021. Discrepancy Detection and Incremental Detection. *PVLDB* 14, 8 (2021), 1351–1364.
[48] Wenfei Fan, Resul Tugay, Yaoshu Wang, Min Xie, and Muhammad Asif Ali. 2023. Learning and Deducing Temporal Orders. *PVLDB* 16, 8 (2023), 1944–1957.
[49] Lior Friedman and Shaul Markovitch. 2018. Recursive feature generation for knowledge-based learning. *arXiv preprint arXiv:1802.00050* (2018).
[50] 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.
[51] 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.
[52] Michael Garey and David Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company.
[53] 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.
[54] Mark A. Hall. 2000. Correlation-based Feature Selection for Discrete and Numeric Class Machine Learning. In *International Conference on Machine Learning (ICML)*. Morgan Kaufmann, 359–366.
[55] Asaf Harari and Gilad Katz. 2022. Automatic features generation and selection from external sources: A DBpedia use case. *Information Sciences* 582 (2022), 398–414.
[56] Asaf Harari and Gilad Katz. 2022. Few-Shot Tabular Data Enrichment Using Fine-Tuned Transformer Architectures. In *ACL*. Association for Computational Linguistics, 1577–1591.
[57] Benjamin Hilprecht and Carsten Binnig. 2021. ReStore - Neural Data Completion for Relational Databases. In *SIGMOD*. 710–722.
[58] Arthur E. Hoerl and Robert W. Kennard. 2000. Ridge Regression: Biased Estimation for Nonorthogonal Problems. *Technometrics* 42, 1 (2000), 80–86.
[59] 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.
[60] Boyi Hou, Qun Chen, Yanyan Wang, Youcef Nafa, and Zhanhuai Li. 2022. Gradual Machine Learning for Entity Resolution. *TKDE* 34, 4 (2022), 1803–1814.
[61] 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).
- [62] Robert Isele, Anja Jentzsch, and Christian Bizer. 2010. Silk server-adding missing links while consuming linked data. In *COLD*. 85–96.
- [63] 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.
- [64] Edward G. Coffman Jr. and Ravi Sethi. 1976. A generalized bound on LPT sequencing. In *SIGMETRICS*. ACM, 306–310.
- [65] 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.
- [66] 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.
- [67] 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.
- [68] 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.
- [69] Ron Kohavi and George H. John. 1997. Wrappers for Feature Subset Selection. *Artif. Intell.* 97, 1-2 (1997), 273–324.
- [70] loannis Koumarelas, Thorsten Papenbrock, and Felix Naumann. 2020. MDedup: Duplicate Detection with Matching Dependencies. *PVLDB* 13, 5 (2020), 712–725.
- [71] Walter Kropatsch. 1996. Building irregular pyramids by dual-graph contraction. In *Vision Image and Signal Processing*.
- [72] Clyde P. Kruskal, Larry Rudolph, and Marc Snir. 1990. A complexity theory of efficient parallel algorithms. *Theoretical Computer Science* 71, 1 (1990), 95–132.
- [73] 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.
- [74] 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).*
- [75] Bing Li, Yukai Miao, Yaoshu Wang, Yifang Sun, and Wei Wang. 2021. Improving the Efficiency and Effectiveness for BERT-based Entity Resolution. In *AAAI*.
- [76] 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.
- [77] 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.
- [78] 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.
- [79] Jiabin Liu, Chengliang Chai, Yuyu Luo, Yin Lou, Jianhua Feng, and Nan Tang. 2022. Feature Augmentation with Reinforcement Learning. In *ICDE*. IEEE, 3360–3372.
- [80] 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).
- [81] Mohammad Mahdavi and Ziawasch Abedjan. 2020. Baran: Effective Error Correction via a Unified Context Representation and Transfer Learning. *PVLDB* 13, 11 (2020), 1948–1961.
- [82] Stephan Mertens. 2006. The Easiest Hard Problem: Number Partitioning. In *Computational Complexity and Statistical Physics*, Allon G. Percus, Gabriel Istrate, and Cristopher Moore (Eds.). Oxford University Press, 125–140.
- [83] 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.
- [84] 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.
- [85] 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.
- [86] 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.
- [87] Avanika Narayan, Ines Chami, Laurel J. Orr, and Christopher Ré. 2022. Can Foundation Models Wrangle Your Data? *PVLDB* 16, 4 (2022), 738–746.
- [88] Fatemeh Nargesian, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. 2018. Table Union Search on Open Data. *PVLDB* 11, 7 (2018), 813–825.
- [89] Daniel Obraczka, Jonathan Schuchart, and Erhard Rahm. 2021. EAGER: Embedding-Assisted Entity Resolution for Knowledge Graphs. *CoRR* abs/2101.06126 (2021).
- [90] 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.
- [91] Jan Peters and J. Andrew Bagnell. 2017. Policy Gradient Methods. In *Encyclopedia of Machine Learning and Data Mining*. Springer.
- [92] Abdulhakim Qahtan, Nan Tang, Mourad Ouzzani, Yang Cao, and Michael Stonebraker. 2020. Pattern functional dependencies for data cleaning. *PVLDB* 13, 5 (2020), 684–697.
- [93] 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.
- [94] Kun Qian, Lucian Popa, and Prithviraj Sen. 2017. Active Learning for Large-Scale Entity Resolution. In *CIKM*. 1379–1388.
- [95] 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.
- [96] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Conference on Empirical Methods in Natural Language Processing*.
- [97] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *PVLDB* 10, 11 (2017), 1190–1201.
- [98] Aécio Santos, Aline Bessa, Fernando Chirigati, Christopher Musco, and Juliana Freire. 2021. Correlation sketches for approximate join-correlation queries. In *SIGMOD*. 1531–1544.
- [99] 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.
- [100] 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).
- [101] 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).
- [102] Feichen Shen and Yugyung Lee. 2016. Knowledge discovery from biomedical ontologies in cross domains. *PloS one* 11, 8 (2016), e0160005.
- [103] 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.
- [104] 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.
- [105] 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.
- [106] Indro Spinelli, Simone Scardapane, and Aurelio Uncini. 2020. Missing data imputation with adversarially-trained graph convolutional networks. *Neural Networks* (2020).
- [107] 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.
- [108] R. Tibshirani. 1996. Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society (Series B)* 58 (1996), 267–288.
- [109] 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.
- [110] Jianhong Tu, Xiaoyue Han, Ju Fan, Nan Tang, Chengliang Chai, Guoliang Li, and Xiaoyong Du. 2022. DADER: Hands-Off Entity Resolution with Domain Adaptation. *PVLDB* 15, 12 (2022), 3666–3669.
- [111] 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.
- [112] 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.
- [113] 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.
- [114] Yue Wang and Shaofeng Zou. 2022. Policy Gradient Method For Robust Reinforcement Learning. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA.*

[115] Melanie Weis and Felix Naumann. 2005. DogmatiX Tracks down Duplicates in XML. In *SIGMOD*. ACM, 431–442.

[116] Steven Euijong Whang and Hector Garcia-Molina. 2013. Joint entity resolution on multiple datasets. *VLDB J.* 22, 6 (2013), 773–795.

[117] Richard Wu, Aoqian Zhang, Ihab F. Ilyas, and Theodoros Rekatsinas. 2020. Attention-based Learning for Missing Data Imputation in HoloClean. In *MLSys 2020*.

[118] Jinsung Yoon, James Jordon, and Mihaela van der Schaar. 2018. GAIN: Missing Data Imputation using Generative Adversarial Nets. In *ICML*. PMLR, 5675–5684.

[119] 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.

[120] Reza Zafarani and Huan Liu. 2016. Users joining multiple sites: Friendship and popularity variations across sites. *Inf. Fusion* 28 (2016), 83–89.

[121] 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*.

[122] Yi Zhang and Zachary G Ives. 2020. Finding related tables in data lakes for interactive data science. In *SIGMOD*. ACM, 1951–1966.

[123] 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).

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

[125] 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.

[126] Barret Zoph and Quoc V. Le. 2017. Neural Architecture Search with Reinforcement Learning. In *ICLR*. OpenReview.net. https://openreview.net/forum?id=r1Ue8Hcxg

[127] 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).

[128] 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.

APPENDIX
PROOF OF THEOREM 1

Proof: We show that schema enrichment is NP-hard by reduction from X3C (Exact Cover by 3-sets), which is NP-complete (cf. [52]). We consider two typical methods for ER and HER: (1) rule-based, and (2) ML-based. Below we give reductions for the two methods.

(1) Proof for rule-based methods. We consider rule-based methods that have the following features, as commonly found in practice.

(1) To conduct ER, $\mathcal{A}_{\text{ER}}$ adopts a key-based strategy. For a schema R there exist multiple sets $\mathcal{K}_1, \mathcal{K}_2, \ldots, \mathcal{K}_n$ of attributes in R that form keys for instances D of R. That is, for any tuples t_1 and t_2 in D, $\mathcal{A}_{\text{ER}}$ returns true (*i.e.,* $\mathcal{A}_{\text{ER}}(t_1, t_2)$ = true) only when there exists a set $\mathcal{K}_j = \{A_1, \ldots, A_t\}$ ($j \in [1, n]$) such that the attributes $A_1, \ldots, A_t$ of t_1 and t_2 are pairwise similar, based on common similarity predicates used in, *e.g.,* Matching Dependencies (MDs) [43].

Many rule-based methods adopt keys one way or another, *e.g.,* MDs, MRLs [32] and unique constraints [53]).

The keys of enriched relation D_G are extended from the keys defined on D using at least one extracted attribute, since we expect the extended attributes to improve the F_1 of $\mathcal{A}_{\text{ER}}$ on D via D_G.

(2) There are infinite tuples of schema R such that for each number n and each tuple t, there exist $2n$ tuples $t_1, \ldots, t_n$ and $t'_1, \ldots, t'_n$ such that (a) $t_1, \ldots, t_n$ match t and (b) $t'_1, \ldots, t'_n$ do not match t; *i.e.,* $\mathcal{A}_{\text{ER}}(t, t_i)$ = true and $\mathcal{A}_{\text{ER}}(t, t'_i)$ = false ($i \in [1, n]$).

(3) The HER mapping is also key-based, *i.e.,* there exist attributes $A_1, \ldots, A_t$ of a tuple t in D and properties $B_1, \ldots, B_t$ of a vertex v in G such that t in D matches v when attribute A_i of t and property B_i of v are similar for all $i \in [1, t]$. Here property B_i may be linked by a path from v, rather than an attribute of v.

Many HER mappings are essentially key-based, *e.g.,* [41, 90]. Indeed, the methods either (a) extract graph properties, store the graph data as relations and conduct ER using key-based methods, *e.g.,* [25, 32] for entities across multiple tables, and [43, 53] for entities in the same table, or (b) convert relation D into a graph and conduct ER on graphs using rules defined on graphs, *e.g.,* GEDs [45], PG-Keys [13], or inductive property matching [41].

Reduction. Next we show that the schema enrichment problem is NP-hard by reduction from X3C. X3C is to decide, given a set $S = \{e_1, e_2, \ldots, e_{3q}\}$ of elements with $|S| = 3q$ and a collection $C = \{S_1, \ldots, S_n\}$ of 3-element subsets of S (*i.e.,* $S_i \subseteq S$ and $|S_i| = 3$), whether there exists an exact cover of S, *i.e.,* a sub-collection $C' \subseteq C$ such that each element $e_j \in S$ is in exactly one set $S_i \in C'$ [52].

Given the set S and the collection C, we construct a relation schema R, a relation D of schema $\mathcal{R}$, a knowledge graph G, a positive integer m and an improvement threshold σ such that S has an exact cover if and only if the improvement of ER on D via D_G is no less than σ. Intuitively, we (1) use $3q$ tuples $t_1, \ldots, t_{3q}$ in D and $3q$ vertices $v_1, \ldots, v_{3q}$ in G to represent all elements in S; (2) n edges in G to represent subsets in C, such that if an element e_i is in a subset S_j (*i.e.,* $e_i \in S_j$), then vertex v_i has an edge (*i.e.,* property) labeled P_j; (3) to match tuples in D with vertices in G, we introduce an attribute A_2 in both D and G such that tuple t_i and vertex v_i that represent the same entity carry the same value N_i; in this way, we can ensure that each vertex $v \in S$ can match only one tuple t in D, and then improve the F_1 of $\mathcal{A}$ by extracting properties from G (see below); (4) to maximize the improvement of ER on D, we also introduce $3q$ new tuples $t'_1, \ldots, t'_{3q}$ and $3q$ vertices $v'_1, \ldots, v'_{3q}$ in G, to represent new entities that are different from the ones represented by $t_1, \ldots, t_{3q}$; (5) intuitively, tuples $t_1, \ldots, t_{3q}$ correspond to one entity α_1, and the newly added tuples $t'_1, \ldots, t'_{3q}$ correspond to a different entity α_2, by referencing the "complete relation D_c", *i.e.,* the instance of a "complete schema" $R_c = (\bar{A}, \bar{C})$, where $\bar{A}$ is the set of attributes in R, $\bar{C}$ is a set of additional attributes, and $\bar{A}$ and $\bar{C}$ cover all necessary attributes of entities in $\mathcal{E}$ (Section 3.1) for ER; and (6) the number m of extracted attributes is set to be q, and the threshold σ for the improvement is $1/3$.

This completes the construction. One can verify that before conducting the enrichment, the F_1 of $\mathcal{A}_{\text{ER}}$ on D is $2/3$. After the enrichment, the improvement can be assessed as follows.

(a) If there exists an exact cover, we can completely separate tuples $t_1, \ldots, t_{3q}$ from $t'_1, \ldots, t'_{3q}$, and then the F_1 of $\mathcal{A}_{\text{ER}}$ on D via D_G reaches 1, *i.e.,* the improvement reaches the threshold $\sigma = 1/3$.

(b) If there exist no exact cover, at least one of the tuples $t_1, \ldots, t_{3q}$ cannot be distinguished from $t'_1, \ldots, t'_{3q}$; then the F_1 of $\mathcal{A}_{\text{ER}}$ on D_G is at most $6q/(6q+1)$ and the improvement of ER on D via D_G is smaller than the threshold $\sigma = 1/3$.

Construction. We define the relation schema R, relation D of $\mathcal{R}$, graph G, integer m and improvement threshold σ as follows.

(1) The relation schema R has two attributes A_1 and A_2 to encode elements in S. More specifically, (a) attribute A_1 is the key used by $\mathcal{A}_{\text{ER}}$; we will construct relation D such that (i) all tuples in D have similar values for A_1, *i.e.,* $\mathcal{A}_{\text{ER}}$ cannot distinguish all tuples in D, but (ii) these tuples can be separated by $\mathcal{A}_{\text{ER}}$ using the enriched relation D_G (see below); and (b) attribute A_2 is used to link tuples in D and vertices in G, *i.e.,* it is the key of the HER mapping.

(2) As shown in Figure 6, the relation D of R consists of $6q$ tuples, two for each element in S. More specifically, for each element $e_i \in S$, D carries two tuples t_i and t'_i: (a) $t_i.A_1 = c_i$ and $t'_i.A_1 = c'_i$ for two constants c_i and c'_i; and (b) $t_i.A_2 = N_i$ and $t'_i.A_2 = N'_i$ for another two constants N_i and N'_i. The values $c_1, \ldots, c_{3q}, c'_1, \ldots, c'_{3q}$ are pairwise similar, *i.e.,* they represent the same entity; but $N_1, \ldots, N_{3q}, N'_1, \ldots, N'_{3q}$ are pairwise dissimilar; such values exist by *feature (2)* above. Then $\mathcal{A}_{\text{ER}}$ cannot distinguish t_i and t'_i, *i.e.,* $\mathcal{A}_{\text{ER}}(t_1, t'_1)$ = true *by feature (1)* above; since A_1 is the key adopted by $\mathcal{A}_{\text{ER}}$; note that attribute A_2 is not the key of $\mathcal{A}_{\text{ER}}$.

(3) As shown in Figure 6, graph G is constructed similarly, except that we introduce two distinct entities (or strings) α_1 and α_2 such that $\mathcal{A}_{\text{ER}}(\alpha_1, \alpha_2)$ = false; by *feature (2)*, such distinct entity pairs always exist. The graph G consists of $6q$ subgraphs, such that G_i^1 and G_i^2 are designated for each element e_i ($i \in [1, 3q]$).

The subgraphs G_i^1 and G_i^2 are constructed as follows:

(3.1) Subgraph G_i^1 consists of a star and several isolated vertices; the star represents the relationships between element e_i and the n subsets in C. More specifically, (i) G_i^1 consists of $2n+2$ vertices $V_i = \{v_i, v_{ai}\} \cup V_{\alpha_1} \cup V_{\alpha_2}$; here v_i represents element e_i, v_{ai} encodes A_2 at-

	D	
	$\mathbf{A}_1$	$\mathbf{A}_2$
t_1	c_1	N_1
$\dots$		
t_{3q}	c_{3q}	N_{3q}
t'_1	c'_1	N'_1
$\dots$		
t'_{3q}	c'_{3q}	N'_{3q}

	D_G		
	$\mathbf{A}_1$	$\mathbf{A}_2$	$\mathbf{S}'_1$
t_1	c_1	N_1	α_1
$\dots$			
t_{3q}	c_{3q}	N_{3q}	α_1
t'_1	c'_1	N'_1	α_2
$\dots$			
t'_{3q}	c'_{3q}	N'_{3q}	α_2

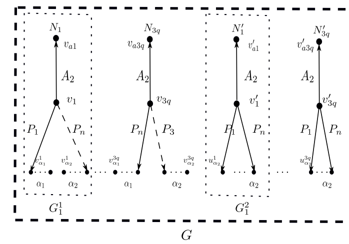


Figure 6: The relations D, D_G and the graph G in the reduction

tribute of v_i, and both V_{α_1} and V_{α_2} consist of n vertices, one for each subset in C; (b) the labels of these vertices are defined as follows: vertices v_i and v_{ai} are labeled c_i and N_i, respectively; and vertices in V_{α_1} (resp. V_{φ_2}) are labeled α_1 (resp. α_2); (c) the edges of G_i^1 are defined as follows: if an element e_i is contained in a subset S_k, then G_i^1 contains an edge $(v_i, v^i_{\alpha_{1k}})$ that is labeled P_k; if an element e_i *is not in* subset S_k, then G contains an edge $(v_i, v^i_{\alpha_{2k}})$ that is also labeled P_k (marked by the dashed line), and vertex v_i also has an edge (v_i, v_{ai}) that is labeled A_2; this edge is used to link vertex v_i to tuple t_i in D via HER; in this way, for each vertex $v \in S_v$ there exists only one tuple t in D that matches v via HER by *feature (3)*; indeed, given a constant N_i we can identify a unique tuple in D that contains the constant N_i.

(3.2) Subgraph G_i^2 is the same as G_i^1 except that (a) vertices v'_i and v'_{ai} are labeled c'_i and N'_i (instead of c_i and N_i), respectively; (b) v'_i has n edges that are linked to vertices labeled α_2 and are labeled $P_1, \dots, P_n$, respectively; and (c) v'_i *has no* edge to a vertex labeled α_1.

This completes the construction of G. Observe that (a) since G consists of stars, the key properties of $v_1, \dots, v_{3q}, v'_1, \dots, v'_{3q}$ are all linked via edges, instead of paths; (b) labels $P_1, \dots, P_n$ do not exist in the relation D, and we can use $P_1, \dots, P_n$ as the extracted attributes; and (c) due to the difference between G_i^1 and G_i^2 (*e.g.,* G_i^2 does not have vertices labeled α_1), we can separate tuples t_i and t'_i, and then improve the F_1 of $\mathcal{A}_{\mathsf{ER}}$ on D (see below).

(4) We define the parameter m to be q ($m = q$) and the threshold σ to be 1/3. Note that extracting m attributes corresponds to picking m subsets from the collection C. One can verify that after extracting m attributes (*i.e.,* picking q subsets), the improvement is at least σ if and only if there exists an exact cover of S (see below).

(5) To compute the F_1, we make use of the following: (a) tuples $t_1, \dots, t_{3q}$ represent the same entity α_1, (b) $t'_1, \dots, t'_{3q}$ represent another entity α_2, and (c) α_1 and α_2 are two distinct entities.

Then we can compute the F_1 of $\mathcal{A}_{\mathsf{ER}}$ on D before the schema enrichment as follows. Note that tuples in D have similar values for the key attribute A_1, and all tuples in D are matched via $\mathcal{A}_{\mathsf{ER}}$ by *feature (2)*. Then Precision and Recall of $\mathcal{A}_{\mathsf{ER}}$ on D are 1/2 and 1, respectively. Therefore, the F_1 of $\mathcal{A}_{\mathsf{ER}}$ on D is 2/3.

After the enrichment, the F_1 of $\mathcal{A}_{\mathsf{ER}}$ on D_G (*i.e.,* the enriched relation) can be computed as follows.

(a) When there exists an exact cover $\{S_{i1}, \dots, S_{iq}\}$ of S, we can enrich D such that the values for the extracted attributes of $t_1, \dots, t_{3q}$ (*i.e.,* the attributes $P_{i1}, \dots, P_{iq}$) are α_1, while the extracted attributes of $t'_1, \dots, t'_{3q}$ are α_2. Then $\mathcal{A}_{\mathsf{ER}}$ can distinguish tuples corresponding to entities α_1 and α_2 (*i.e.,* $\mathcal{A}_{\mathsf{ER}}(t_i, t'_j)$ = false; that is, tuples t_i and t'_j are distinguished with $i, j \in [1, 3q]$), since the keys on D_G use at least one extracted attribute; but $\mathcal{A}_{\mathsf{ER}}$ cannot distinguish tuples among $t_1, \dots, t_{3q}$ or tuples among $t'_1, \dots, t'_{3q}$, since they have similar values for the key attribute A_1. Then both Precision and Recall of method $\mathcal{A}_{\mathsf{ER}}$ on D_G are 1. Hence the F_1 of $\mathcal{A}_{\mathsf{ER}}$ on D_G is 1, *i.e.,* the accuracy improvement reaches 1/3.

(b) When there does not exist an exact cover, we cannot distinguish at least one element t_i from $t'_1, \dots, t'_{3q}$. Hence one can verify that Precision and Recall of $\mathcal{A}_{\mathsf{ER}}$ on D_G are at most $6q/(6q+1)$ and 1, respectively. Hence, the F_1 of $\mathcal{A}_{\mathsf{ER}}$ on D_G is at most $6q/(6q+1)$, and the accuracy improvement is below 1/3.

This completes the construction. One can verify that the reduction is in PTIME, as there must exist polynomial sizes of constants $c_1, \dots, c_{3q}, c'_1, \dots, c'_{3q}, N_1, \dots, N_{3q}, N'_1, \dots, N'_{3q}$, α_1 and α_2.

Correctness. We next show that S has an exact cover if and only if the improvement of ER on D via D_G is at least $\sigma = 1/3$.

($\Rightarrow$) Assume that S has an exact cover $S_{i1}, \dots, S_{iq}$. We enrich D with attributes $P_{i1}, \dots, P_{iq}$, and show that the improvement of ER on D via D_G is at least σ. By the definition of G, the attributes P_{ik} ($k \in [1, q]$) of tuples $t_1, \dots, t_{3q}$ are α_1, while the attributes P_{ik} ($k \in [1, q]$) of tuples $t'_1, \dots, t'_{3q}$ are α_2. We can verify that (a) t_i and t'_j are not matched via $\mathcal{A}_{\mathsf{ER}}$, *i.e.,* $\mathcal{A}_{\mathsf{ER}}(t_i, t'_j)$ = false, by *feature (1)*; indeed, the P_{ik} attributes of t_i and t'_j are distinct (α_1 vs. α_2); moreover, (b) $\mathcal{A}_{\mathsf{ER}}$ cannot distinguish tuples $t_1, \dots, t_{3q}$, *i.e.,* $\mathcal{A}_{\mathsf{ER}}(t_i, t_j)$ = true for each $i, j \in [1, 3q]$, by *feature (1)*; similarly for tuples $t'_1, \dots, t'_{3q}$. Thus the improvement of ER on D via D_G is $1/3 = \delta$.

($\Leftarrow$) Assume that the improvement of ER on D via D_G is at least σ. We can obtain an exact cover of S as follows. Since D already has the attribute A_2, the extracted attributes must be among $P_1, \dots, P_n$. Since $m = q$, assume that the extracted attributes are $P_{i1}, \dots, P_{iq}$.

We show that $S_{i1}, \dots, S_{iq}$ form an exact cover of S. It suffices to prove that each element $e_i \in S$ exists in some S_{ij} with $j \in [1, q]$. If it holds, since (1) each S_{ij} consists of 3 elements, (2) there exist q sets $S_{i1}, \dots, S_{iq}$ and (3) $|S| = 3q$, each element $e_i \in S$ is in exact one S_{ij}. Therefore, $S_{i1}, \dots, S_{iq}$ form an exact cover of S. We show that each element $e_i \in S$ exists in some S_{ij} with $j \in [1, q]$ by contradiction. Assume that $S_{i1}, \dots, S_{iq}$ do not form an exact cover; then there exists at least one element $e_i \in S$ such that e_i is not contained in $S_{i1}, \dots, S_{iq}$. From the construction of G, we know that the attributes $P_{i1}, \dots, P_{iq}$ of the tuple t_i are all α_2 (marked by the dashed lines in Figure 6), where t_i is the tuple in D_G that represents the element e_i (*i.e.,* $t_i.A_2 = N_i$); indeed, since e_i is not contained in $S_{i1}, \dots, S_{iq}$, edges labeled $P_{i1}, \dots, P_{iq}$ of the vertex v_i are linked to vertex labeled α_2 in G; here v_i is the vertex in G that represents the element e_i. Then $\mathcal{A}_{\text{ER}}$ cannot distinguish t_i from tuples $t'_1, \ldots, t'_{3q}$ in D_G (*i.e.,* $\mathcal{A}_{\text{ER}}(t_i, t'_j) = \text{true}$) by *feature (1) given above.* Then Precision and Recall of $\mathcal{A}_{\text{ER}}$ on D_G are at most $6q/(6q+1)$ and 1, respectively. Therefore, the F_1 of $\mathcal{A}_{\text{ER}}$ on D_G is at most $6q/(6q+1)$ and the improvement is $6q/(6q+1) - 2/3 < 1 - 2/3 = 1/3$ and is below the threshold $\sigma = 1/3$, a contradiction. □

(2) Proof for ML-based methods. We show that schema extraction is NP-hard by slightly extending the reduction above when ER and HER are ML-based. Recall three features for rule-based methods. To reuse the reduction, we revise these features for ML-based methods. More specifically, we assume the following.

(1) Assume that the $\mathcal{A}_{\text{ER}}$ algorithm inspects some key attributes, and conducts ER by aggregating value similarity of these key attributes. This assumption is satisfied by most ML-based ER methods in practice, such as DeepMatcher [86] and Ditto [77]; for example, DeepMatcher first sums up the attribute similarities, and then normalizes the output for further comparison [86].

To enforce the assumption, we further assume that these ML models are selective, *i.e.,* if the values of one key attribute of two tuples are dissimilar, then they correspond to different entities; this can be achieved by setting a high threshold for these models [60].

On the enriched relation D_G, $\mathcal{A}_{\text{ER}}$ not only inspects the original key attributes in D but also at least one extracted attribute.

(2) We also assume that there are infinitely many tuples of schema R. This is the same as the one for the rule-based methods. The assumption is also satisfied by most ML-based ER algorithms [77, 86].

(3) The ML-based HER mappings are also selective; *i.e.,* a tuple t in D and a vertex v in G match, only when all key attributes $A_1, \ldots, A_t$ of t and key properties $B_1, \ldots, B_t$ of v are pairwise similar, *i.e.,* $t.A_i$ and $v.B_i$ are similar for all $i \in [1, t]$. Note that (a) there may exist multiple sets of key attributes for t and v; (b) attribute A_i and B_i ($i \in [1, t]$) may bear different labels; and (c) the properties $B_1, \ldots, B_t$ of v may be linked to v via simple paths as mentioned above.

This assumption is satisfied by most algorithms for HER, *e.g.,* (a) DeepMatcher [86] and Ditto [77] when graphs are stored in relations; and (b) parametric simulation [41], EAGER [89] and GBC-ER [24] when relations are converted to a graph. As mentioned above, when the thresholds of these algorithms are high, if tuple t in D and vertex v in G do not have similar values (*i.e.,* different entities) for the same key attribute, then they represent different entities.

Under these assumptions, we can slightly revise the NP-hardness proof above for ML-based methods. More specifically, we replace constants $c_1, \ldots, c_{3q}, c'_1, \ldots, c'_{3q}$, $N_1, \ldots, N_{3q}, N'_1, \ldots, N'_{3q}$, α_1 and α_2 with new constants that are recognized by ML-based ER algorithms $\mathcal{A}_{\text{ER}}$ and HER mappings, since these constants are selected to ensure that tuples in D are predicted to be the same or different tuples via the $\mathcal{A}_{\text{ER}}$ algorithm and the HER mapping (see below), and different algorithms demand different constants.

We can easily verify the correctness of the revised reduction as above. Indeed, these constants ensure that (1) $\mathcal{A}_{\text{ER}}$ cannot distinguish tuple t_i from t'_j ($i, j \in 3q$) in D, and (2) the HER mapping maps tuple t_i in D to vertex v_i in G and tuple t'_i to vertex v'_i. Since we replace these constants with new constants that are recognized by ML-based ER algorithms $\mathcal{A}_{\text{ER}}$ and HER mappings, $\mathcal{A}_{\text{ER}}$ and the HER mapping return the same results as the ones in the above reduction. Therefore, we can similarly show that S has an exact cover if and only if the improvement of ER on D via D_G is at least $\sigma = 1/3$. □

PARALLEL DATA ENRICHMENT

We next parallelize online algorithms BEnrich and IncEnrich.

Parallel batch enrichment

We parallelize algorithm BEnrich to PBEnrich, which employs a coordinator P_0 and a set of n workers $P_1, \ldots, P_n$, where the coordinator P_0 is responsible for distributing and balancing workloads, and the n workers conduct enrichment in parallel.

Overview. PBEnrich conducts HER matching and populating as follows. Coordinator P_0 first estimates the enrichment cost, based on which it constructs n workloads $\mathcal{W}_1, \ldots, \mathcal{W}_n$, and distributes them to $P_1, \ldots, P_n$. Each worker P_i ($i \in [1, n]$) runs BEnrich on its workload $\mathcal{W}_i$ locally, and all workers run in parallel. Finally, all workers send their results to P_0, and P_0 assembles the enriched relation D_G. Here to simplify the discussion, we assume that G is replicated at each worker. It can be extended to handle G fragmented across workers, by fetching relevant data when necessary.

Below we present the details of each of these steps.

Cost estimation. As shown in BEnrich, the enrichment of different tuples is independent of each other. Thus, we construct a work unit $w(t)$ for each tuple t in D. Its cost, denoted by $\text{cost}(w(t))$, is dominated by HER matching which retrieves the candidate matches and populating which traverses the paths from top-K matching vertices and assigns the top-ranked value via the ranking model. Thus, we measure $\text{cost}(w(t))$ by the (weighted) sum of $|C_t|$ (*i.e.,* the number of candidate matches after blocking) and $\sum_{v \in \mathcal{V}_t} d_v$, where $\mathcal{V}_t$ is the set of top-K HER matches and d_v is the degree of v; intuitively, a vertex with a higher degree is a hub vertex that typically has more candidate values to be considered. Given a set D_i of tuples, the workload $\mathcal{W}_i$ is $\{w(t) \mid t \in D_i\}$ and the enrichment cost of $\mathcal{W}_i$ is computed to be $\text{cost}(\mathcal{W}_i) = \sum_{t \in D_i} \text{cost}(w(t))$.

Workload balancing. We partition D into n sets of tuples $D_1, \ldots, D_n$, and construct n workloads $\mathcal{W}_1, \ldots, \mathcal{W}_n$, where $\mathcal{W}_i$ is sent to worker P_i. The workloads *w.l.o.g.* different D_i's are evenly partitioned across workers. To achieve workload balancing, we minimize the maximum enrichment cost among $\{\text{cost}(\mathcal{W}_1), \ldots, \text{cost}(\mathcal{W}_n)\}$. The workload balancing is NP-hard and can be reduced from k-way number partitioning problem [82]. We adopt the well-known greedy number partitioning strategy. More specifically, we sort $\{w(t_i)\}_{i=1}^{|D|}$ in the increasing order, and greedily add each $w(t)$ to $\mathcal{W}_i$ with the smallest $\text{cost}(\mathcal{W}_i)$. The time complexity of this strategy is $O(|D|\log|D|)$ and the approximation ratio is $\frac{4n-1}{3n}$ [64]. To reduce the sorting cost, we can cluster $\{w(t_i)\}_{i=1}^{|D|}$ into L groups and sort them, such that the complexity is reduced to $O(|L|\log|L|)$.

Parallel scalability. To measure the effectiveness of PBEnrich, we adopt the widely used notion of *parallel scalability* [72].

Consider a problem $\mathcal{P}$ on a relation D and a graph G. We denote by $T_s(|I_{\mathcal{P}}|, |D|, |G|)$ the worst-case complexity of a sequential algorithm $\mathcal{A}$ for handling an instance $I_{\mathcal{P}}$ of problem $\mathcal{P}$. For a parallel algorithm $\mathcal{A}_p$ for problem $\mathcal{P}$, we denote by $T_p(|I_\mathcal{P}|, |D|, |G|, n)$ the time taken by it for processing problem instance $I_\mathcal{P}$ using n processors. We say that algorithm $\mathcal{A}_p$ is *parallelly scalable relative to* $\mathcal{A}$ if

$$T_p(|I_\mathcal{P}|, |D|, |G|, n) = O\big(\frac{T_s(|I_\mathcal{P}|, |D|, |G|)}{n}\big)$$

for any instance $I_\mathcal{P}$. That is, the parallel algorithm $\mathcal{A}_p$ is able to "linearly" reduce the sequential cost of a yardstick algorithm $\mathcal{A}$.

Theorem 2: PBEnrich *is parallelly scalable relative to* BEnrich. □

Proof. The communication cost of PBEnrich is $O(|D|)$ since (a) G is replicated at all workers, and (b) the data and auxiliary structures for each work unit are distributed to the workers. This cost is smaller than the computation cost, which is $O(|D_i||G| + |D_i||C_{\max}|Km)$ ($i \in [1, n]$). Moreover, the workload balancing strategy evenly partitions the workloads and guarantees a $\frac{4n-1}{3n}$ approximation ratio. Thus, PBEnrich is parallelly scalable relative to BEnrich. □

Parallel Incremental Enrichment

We parallelize incremental algorithm IncEnrich to PIncEnrich along the same line as PBEnrich. We partition D into n partitions $D_1, \ldots, D_n$ such that worker P_i processes workload $\mathcal{W}_i$ to enrich tuples in D_i in response to ΔG. PIncEnrich adopts cost estimation and workload balancing similar to those in PBEnrich.

Theorem 3: IncEnrich *is parallelly scalable to* IncEnrich. □

Proof. The cost of PIncEnrich is $O(c_{\text{up}} m \#\text{Aff}_i |\Delta G||P|)$, where $\#\text{Aff}_i$ is the maximum number of tuples in D_i affected by ΔG from one path match. Similar to Theorem 2, one can show that the computation dominates the complexity, which is evenly distributed across workers. IncEnrich is parallelly scalable relative to IncEnrich. □