

Crowdsourced Collective Entity Resolution with Relational Match Propagation

Jiacheng Huang[†], Wei Hu^{†*}, Zhifeng Bao[‡] and Yuzhong Qu[†]

[†]State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China

[‡]RMIT University, Melbourne, Australia

Email: jchuang.nju@gmail.com, wuhu@nju.edu.cn, zhifeng.bao@rmit.edu.au, yzqu@nju.edu.cn

Abstract—Knowledge bases (KBs) store rich yet heterogeneous entities and facts. Entity resolution (ER) aims to identify entities in KBs which refer to the same real-world object. Recent studies have shown significant benefits of involving humans in the loop of ER. They often resolve entities with pairwise similarity measures over attribute values and resort to the crowds to label uncertain ones. However, existing methods still suffer from high labor costs and insufficient labeling to some extent. In this paper, we propose a novel approach called crowdsourced collective ER, which leverages the relationships between entities to infer matches jointly rather than independently. Specifically, it iteratively asks human workers to label picked entity pairs and propagates the labeling information to their neighbors in distance. During this process, we address the problems of candidate entity pruning, probabilistic propagation, optimal question selection and error-tolerant truth inference. Our experiments on real-world datasets demonstrate that, compared with state-of-the-art methods, our approach achieves superior accuracy with much less labeling.

I. INTRODUCTION

Knowledge bases (KBs) store rich yet heterogeneous entities and facts about the real world, where each fact is structured as a triple in the form of $(entity, property, value)$. Entity resolution (ER) aims at identifying entities referring to the same real-world object, which is critical in cleansing and integration of KBs. Existing approaches exploit diversified features of KBs, such as attribute values and entity relationships, see surveys [1], [2], [3], [4]. Recent studies have demonstrated that *crowdsourced ER*, which recruits human workers to solve micro-tasks (e.g., judging if a pair of entities is a match), can improve the overall accuracy.

Current crowdsourced ER approaches mainly leverage *transitivity* [5], [6], [7] or *monotonicity* [8], [9], [10], [11], [12] as their resolution basis. The transitivity-based approaches rely on the observation that the match relation is usually an *equivalence* relation. The monotonicity-based ones assume that each pair of entities can be represented by a similarity vector of attribute values, and the binary classification function, which judges whether a similarity vector is a match, is monotonic in terms of the *partial order* among the similarity vectors.

However, both kinds of approaches can hardly infer matches across different types of entities. Let us see Figure 1 for example. The figure shows a directed graph, called *entity resolution graph* (ER graph), in which each vertex denotes a pair of entities and each edge denotes a relationship between

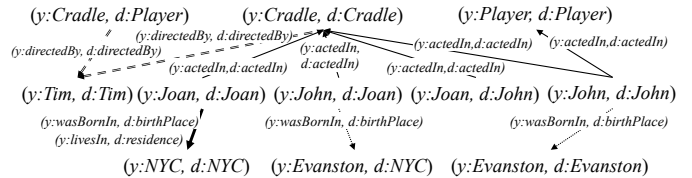


Fig. 1: An ER graph example between YAGO and DBpedia. Entities in YAGO are prefixed by “y:”, and entities in DBpedia are prefixed by “d:”. *Joan*, *John* and *Tim* are persons. *Cradle* and *Player* are movies. *NYC* and *Evanston* are cities.

two entity pairs. Assume that $(y:Joan, d:Joan)$ is labeled as a match, the birth place pair $(y:NYC, d:NYC)$ is expected to be a match. Since these two pairs are in different equivalence classes, the transitivity-based approaches are apparently unable to take effect. As different relationships (like $y:directedBy$ and $y:wasBornIn$) make most similarity vectors of entities of different types incomparable, the monotonicity-based approaches have to handle them separately.

In this paper, we propose a new approach called Remp (Relational match propagation) to address the above problems. The main idea is to leverage *collective ER* that resolves entities connected by relationships jointly and distantly, based on a small amount of labels provided by workers. Specifically, Remp iteratively asks workers to label a few entity pairs and propagates the labeling information to their neighboring entity pairs in distance, which are then resolved jointly rather than independently. There remain two challenges to achieve such a crowdsourced collective ER.

The first challenge is how to conduct an effective relational match propagation. Relationships like functional/inverse functional properties in OWL [13] (e.g., $y:wasBornIn$) provide a strong evidence, but these properties only account for a small portion while the majority of relationships is multi-valued (e.g., $actedIn$). Multi-valued relationships often connect non-matches to matches (e.g., $(y:John, d:Joan)$ is connected to $(y:Cradle, d:Cradle)$ in Figure 1). Therefore, we propose a new relational match propagation model, to decide which neighbors can be safely inferred as matches.

The second challenge is how to select good questions to ask workers. For an ER graph involving two large KBs, the number of vertices (i.e. candidate questions) can be quadratic. We introduce an entity pair pruning algorithm to narrow the search space of questions. Moreover, different questions have

*Corresponding author

different inference power. In order to maximize the expected number of inferred matches, we propose a question selection algorithm, which chooses possible entity matches scattered in different parts of the ER graph to achieve the largest number of inferred matches.

In summary, the main contributions of this paper are listed as follows:

- We design a partial order based entity pruning algorithm, which significantly reduces the size of an ER graph.
- We propose a relational match propagation model, which can jointly infer the matches between different types of entities in distance.
- We formulate the problem of optimal multiple questions selection with cost constraint, and design an efficient algorithm to obtain approximate solutions.
- We present an error-tolerant method to infer truths from imperfect human labeling. Moreover, we train a classifier to handle isolated entity pairs.
- We conduct real-world experiments and comparison with state-of-the-art approaches to assess the performance of our approach. The experimental results show that our approach achieves superior accuracy with much fewer labeling tasks.

Paper organization. Section II reviews the literature. Section III defines the problem and sketches out the approach. In Sections IV–VII, we describe the approach in detail. Section VIII reports the experiments and results. Last, Section IX concludes this paper.

II. RELATED WORK

A. Crowdsourced ER

Inference models. Based on the transitive relation of entity matches, many approaches such as [5], [14] make use of prior match probabilities to decide the order of questions. Firmani et al. [7] proved that the optimal strategy is to ask questions in descending order of entity cluster size. They formulated the problem of crowdsourced ER with early termination and put forward several question ordering strategies. Although the transitive relation can infer matches within each cluster, workers need to check all clusters.

On the other hand, Arasu et al. [8] investigated the monotonicity property among the similarity vectors of entity pairs. Given two similarity thresholds s_1, s_2 and $s_1 \succeq s_2$, we have $\Pr[u_1 \simeq u_2 | s(u_1, u_2) \succeq s_1] \geq \Pr[u_1 \simeq u_2 | s(u_1, u_2) \succeq s_2]$. ALGPR [8] and ERLEARN [11] use the monotonicity property to search new thresholds, and estimate the precision of results. In particular, the partial order based approaches [15], [16], [12] explore similarity thresholds among similarity vectors. Furthermore, POWER [16] groups similarity vectors to reduce the search space. Corleone [9] and Falcon [10] learn random forest classifiers, where each decision tree is equivalent to a similarity vector. However, these approaches are designed for ER with single entity type. To leverage monotonicity on ER between KBs with complex type information, HIKE [12] uses hierarchical agglomerative clustering to

partition entities with similar attributes and relationships, and uses the monotonicity techniques on each entity partition to find matches. Although our approach also uses monotonicity, it only uses monotonicity to prune candidate entity pairs. In addition, our approach allows match inference between different entity types (e.g., from persons to locations) to reduce the labeling efforts.

Question interfaces. Pairwise and multi-item are two common question interfaces. The pairwise interface asks workers to judge whether a pair of entities is a match [7], [17]. Differently, Marcus et al. [18] proposed a multi-item interface to save questions, where each question contains multiple entities to be grouped. Wang et al. [19] minimized the number of multi-item questions on the given entity pair set such that each question contains at most k entities. Waldo [20] is a recent hybrid interface, which optimizes the trade-off between cost and accuracy of the two question interfaces based on task difficulty. The above approaches do not have the inference power and they may generate a large amount of questions.

Quality control. To deal with errors produced by workers, quality control techniques [6], [20], [21] leverage the correlation between matches and workers to find inaccurate labels, and improve the accuracy by asking more questions about uncertain ones. These approaches gain improvement by redundant labeling.

B. Collective ER

In addition to attribute values, collective ER [22], [23], [24], [25] further takes the relationships between entities into account. CMD [26] extends the probabilistic soft logic to learn rules for ontology matching. LMT [27] learns soft logic rules to resolve entities in a familial network. Because learning a probabilistic distribution on large KBs is time-consuming, PARIS [28] and SiGMa [29] implement message passing-style algorithms that obtain seed matches created by hand crafted rules and pass the match messages to their neighbors. However, they do not leverage crowdsourcing to improve the ER accuracy and may encounter the error accumulation problem.

III. APPROACH OVERVIEW

In this section, we present necessary preliminaries to define our problem, followed by a general workflow of our approach. Frequently used notations are summarized in Table I.

A. Preliminaries & Problem Definition

KBs store rich, structured real-world facts. In a KB, each fact is stated in a triple of the form (*entity, property, value*), where *property* can be either an attribute or a relationship, and *value* can be either a literal or another entity. The sets of entities, literals, attributes, relationships and triples are denoted by U, L, A, R and T , respectively. Therefore, a KB is defined as a 5-tuple $\mathcal{K} = (U, L, A, R, T)$. Moreover, attribute triples $T_{attr} \subseteq U \times A \times L$ attach entities with literals, e.g., (*Leonardo da Vinci, birth date, "1452-4-15"*), and relationship triples $T_{rel} \subseteq U \times R \times U$ link entities by relationships, e.g., (*Leonardo da Vinci, works, Mona Lisa*).

TABLE I: Frequently used notations

Notations	Descriptions
\mathcal{K}, u	a KB and an entity
r, a	a relationship, and an attribute
N_u^r, N_u^a	the value sets of r and a w.r.t. u
p, q	an entity pair, and a question
m_p, m_q	the event that p and q is a match
M	a set of entity or attribute matches
C	a set of candidate questions
Q	a set of asked questions
H	a set of labels

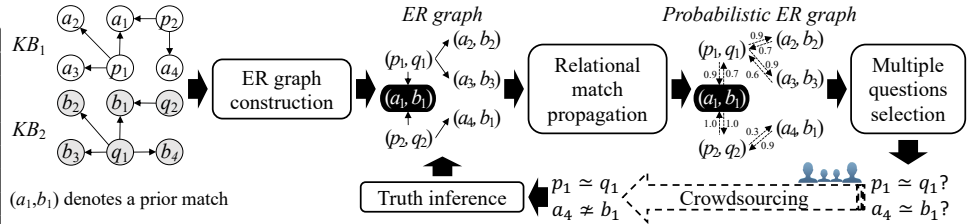


Fig. 2: Workflow of the proposed approach

Entity Resolution (ER) aims to resolve entities in KBs denoting the same real-world thing. Let u_1, u_2 denote two entities in two different KBs. We call the entity pair $p = (u_1, u_2)$ a *match* and denote it by $u_1 \simeq u_2$ or m_p if u_1, u_2 refer to the same. In contrast, we call $p = (u_1, u_2)$ a *non-match* and denote it by $u_1 \not\simeq u_2$ if u_1, u_2 refer to two different objects. Both matches and non-matches are regarded as *resolved* entity pairs, and other pairs are regarded as *unresolved*. Traditionally, *reference matches* (i.e., gold standard) are used to evaluate the quality of the ER results, and precision, recall and F1-score are widely-used metrics.

Crowdsourced ER carries out ER with human helps. Usually, it executes several human-machine loops, and in each loop, the machine picks one or several questions to ask workers to label them and updates the ER results in terms of the labels. Due to the monetary cost of human labors, a crowdsourced ER algorithm is expected to ask limited questions while obtaining as many results as possible.

Definition 1 (Crowdsourced Collective ER). *Given two KBs \mathcal{K}_1 and \mathcal{K}_2 , and a budget, the crowdsourced collective ER problem is to maximize recall with a precision restriction by asking humans to label tasks while not exceeding the budget.*

Specifically, we assume that both KBs contain “dense” relationships and focus on using matches obtained from workers to jointly infer matches with relationships.

B. A Workflow of Our Approach

Given two KBs as input, Figure 2 shows the workflow of our approach to crowdsourced collective ER. After iterating four processing stages, the approach returns a set of matches between the two KBs.

- 1) **ER graph construction** aims to construct a small ER graph by reducing the amount of vertices (i.e. entity pairs). It first conducts a similarity measurement to filter out some non-matches. At the same time, it uses some matches obtained from exact matching [12], [29], [30] to calculate the similarities between attributes and find attribute matches. Then, based on the attribute matches, it assembles the similarities between values to similarity vectors, and leverages the natural partial order on the vectors to prune more vertices.
- 2) **Relational match propagation** models how to use matches to infer the match probabilities of unresolved entity pairs in each connected component of the ER

graph. It first uses some matches and maximum likelihood estimation to measure the consistency of relationships. Then, based on the consistency of relationships and the ER graph structure, it computes the conditional match probabilities of unresolved entity pairs given the matches. The conditional match probabilities derive a probabilistic ER graph.

- 3) **Multiple questions selection** selects a set of unresolved entity pairs in the probabilistic ER graph as questions to ask workers. It models the discovery of inferred match set for each question as the all-pairs shortest path problem and uses a graph-based algorithm to solve it. We prove that the multiple questions selection problem is NP-hard and design a greedy algorithm to find the best questions to ask.
- 4) **Truth inference** infers matches based on the results labeled by workers. It first computes the posterior match probabilities of the questions based on the quality of the workers, and then leverages these posterior probabilities to update the (probabilistic) ER graph. Also, for isolated entity pairs, it builds a random forest classifier to avoid asking the workers to check them one by one.

The approach stops asking more questions when there is no unresolved entity pair that can be inferred by relational match propagation.

IV. ER GRAPH CONSTRUCTION

A. ER Graph

Graph structures [31], [32] are widely used to model the resolution states of entity pairs and the relationships between them. For example, Dong et al. [31] proposed dependency graph to model the dependency between similarities of entity pairs. In this paper, we use the notion of *ER graph* to denote this graph structure. Different from dependency graph, each edge in the ER graph is labeled with a pair of relationships from two KBs.

Definition 2 (ER Graph). *Given two KBs $\mathcal{K}_1 = (U_1, L_1, A_1, R_1, T_1)$ and $\mathcal{K}_2 = (U_2, L_2, A_2, R_2, T_2)$, an ER graph on \mathcal{K}_1 and \mathcal{K}_2 is a directed, edge-labeled multigraph $\mathcal{G} = (V, E, l_e)$, such that (1) $V \subseteq U_1 \times U_2$; (2) for each vertex pair $(u_1, u_2), (u'_1, u'_2) \in V$, $((u_1, u_2), (u'_1, u'_2)) \in \mathcal{G} \wedge l_v((u_1, u_2), (u'_1, u'_2)) = (r_1, r_2)$ if and only if $(u_1, r_1, u'_1) \in T_1 \wedge (u_2, r_2, u'_2) \in T_2$.*

Figure 1 illustrates an ER graph fragment built from DBpedia and YAGO. Note that, an entity can occur in multiple vertices, and a relationship can appear in different edge labels. A *probabilistic ER graph* is an ER graph where each edge $((u_1, u_2), (u'_1, u'_2))$ is labeled with a conditional probability $\Pr(u'_1 \simeq u'_2 \mid u_1 \simeq u_2)$. The major challenge of constructing an ER graph is how to significantly reduce the size of the graph while preserving as many potential entity matches as possible.

B. Candidate Entity Match Generation

We conduct a string matching on entity labels (e.g., the values of *rdfs:label*) to generate *candidate* entity matches and regard them as vertices in the ER graph. Specifically, we first normalize entity labels via lowercasing, tokenization, stemming, etc. Then, we leverage the Jaccard coefficient—the size of the intersection divided by the size of the union of two sets—as our similarity measure to compute similarities on the normalized label token sets and follow the previous studies [5], [16], [19] to prune the entity pairs whose similarities are less than a predefined threshold (e.g., 0.3). Although the choice of thresholds is dataset dependent, this process runs fast and largely reduces the amount of non-matches, thus helping the ER approaches scale up. Note that there are many choices on the similarity metric, e.g., Jaccard, cosine, dice and edit distance [33]; our approach can work with any of them and we use Jaccard for illustration purpose only. The set of candidate entity matches is denoted by M_c . Similar to [12], we use the label similarities as prior match probabilities (i.e., $\Pr[m_p]$). More accurate estimation in [6], [7] can be achieved by human labeling.

C. Attribute Matching

In M_c , we refer to the subset of its entities that has exactly the same labels as *initial* entity matches. We leverage them as a priori knowledge for attribute and relationship matching (see Sections IV-C and V-A). Other features, e.g., *owl:sameAs* and inverse functional properties [13], may also be used to infer initial entity matches [34], [30]. Note that we do not directly add initial entity matches in the final ER results, because they may contain errors. The set of initial entity matches is denoted by M_{in} .

For such a set of initial entity matches M_{in} between two KBs $\mathcal{K}_1 = (U_1, L_1, A_1, R_1, T_1)$ and $\mathcal{K}_2 = (U_2, L_2, A_2, R_2, T_2)$, we proceed to define the following attribute similarity to find their attribute matches. For any two attributes $a_1 \in A_1$ and $a_2 \in A_2$, their similarity $\text{sim}(a_1, a_2)$ is defined as the average similarity of their values:

$$\text{sim}_A(a_1, a_2) = \frac{\sum_{(u_1, u_2) \in M_{in}} \text{sim}_L(N_{u_1}^{a_1}, N_{u_2}^{a_2})}{|\{(u_1, u_2) \in M_{in} : N_{u_1}^{a_1} \cup N_{u_2}^{a_2} \neq \emptyset\}|}, \quad (1)$$

where $N_{u_1}^{a_1} = \{l_1 : (u_1, a_1, l_1) \in T_1\}$ and $N_{u_2}^{a_2}$ is defined analogously. sim_L represents an extended Jaccard similarity measure for two sets of literals, which employs an internal literal similarity measure and a threshold to determine two literals being the same when their similarity is not lower than

the threshold [35]. For different types of literals, we use the Jaccard coefficient for strings and the maximum percentage difference for numbers (e.g., integers, floats and dates). The threshold is set to 0.9 to guarantee high precision. We refer interested readers to [36] for more information about attribute matching.

For simplicity, every attribute in one KB is restricted to match at most one attribute in the other KB. This global 1:1 matching constraint is widely used in ontology matching [37], and facilitates our assembling of similarity vectors (later in Section IV-D). The 1:1 attribute matching selection is modeled as the bipartite graph matching problem and solved with the Hungarian algorithm [38] in $O((|A_1| + |A_2|)^2 |A_1| |A_2|)$ time. The set of attribute matches is denoted by M_{at} .

D. Partial Order Based Pruning

Given the candidate entity match set M_c and the attribute match set M_{at} , for each candidate $(u_1, u_2) \in M_c$, we create a similarity vector $\mathbf{s}(u_1, u_2) = (s_1, s_2, \dots, s_{|M_{at}|})$, where s_i is the literal similarity (sim_L) between u_1 and u_2 on the i^{th} attribute match ($1 \leq i \leq |M_{at}|$). As a consequence, a natural partial order exists among the similarity vectors: $\mathbf{s} \succeq \mathbf{s}'$ if and only if $\forall 1 \leq i \leq |M_{at}|, s_i \geq s'_i$. This partial order can be used to determine whether an entity pair is a (non-)match in two ways: (i) an entity pair (u_1, u_2) is a match if there exists an entity pair (u'_1, u'_2) such that (u'_1, u'_2) is a match and $\mathbf{s}(u_1, u_2) \succeq \mathbf{s}(u'_1, u'_2)$; and (ii) (u_1, u_2) is a non-match if there exists (u'_1, u'_2) such that (u'_1, u'_2) is a non-match and $\mathbf{s}(u'_1, u'_2) \succeq \mathbf{s}(u_1, u_2)$.

We incorporate this partial order into a k -nearest neighbor search for further pruning the candidate entity match set M_c . Let us assume that an entity u_1 in one KB has a set of candidate match counterparts $\{u_2^1, u_2^2, \dots, u_2^J\}$ in another KB. The similarity vectors are written as $\mathbf{s}(u_1, u_2^1), \mathbf{s}(u_1, u_2^2), \dots, \mathbf{s}(u_1, u_2^J)$, and we want to determine the top- k in them. Since the partial order is a weak ordering, we count the number of vectors strictly larger than each pair (u_1, u_2^j) ($1 \leq j \leq J$) as its “rank”, i.e., the minimal rank in all possible refined full orders. Note that the counterparts of entities in one entity pair are both considered. So, the worst rank of an entity pair (u_1, u_2) , denoted by $\text{min_rank}(u_1, u_2)$, is

$$\begin{aligned} \text{min_rank}(u_1, u_2) &= \max_{i \in \{1, 2\}} \text{min_rank}_i(u_1, u_2), \\ \text{min_rank}_1(u_1, u_2) &= |\{u'_2 : \mathbf{s}(u_1, u'_2) \succ \mathbf{s}(u_1, u_2)\}|, \\ \text{min_rank}_2(u_1, u_2) &= |\{u'_1 : \mathbf{s}(u'_1, u_2) \succ \mathbf{s}(u_1, u_2)\}|, \end{aligned} \quad (2)$$

where all $(u_1, u_2), (u_1, u'_2), (u'_1, u_2) \in M_c$.

By min_rank , we design a modified k -nearest neighbor algorithm on this partial order (see Algorithm 1). Because the full order among candidate entity matches is unknown, instead of finding the top- k matches directly, we prune the ones that cannot be in top- k . Thus, each entity pair $(u_1, u_2) \in M_c$ such that $\text{min_rank}(u_1, u_2) \geq k$ needs to be pruned. Also, each pair smaller than a pruned pair should be removed based on the partial order to avoid redundant checking, because min_rank of these pairs must be greater than k . The set of retained entity

Algorithm 1: Partial order based pruning

Input: Candidate entity match set M_c , attribute match set M_{at} , threshold k
Output: Retained entity match set M_{rd}

```
1 foreach  $(u_1, u_2) \in M_c$  do pre-compute  $\mathbf{s}(u_1, u_2)$ ;  
2  $M_{rd} \leftarrow \text{PruningInOneWay}(M_c, U_1, k)$ ;  
3  $M_{rd} \leftarrow \text{PruningInOneWay}(M_{rd}, U_2, k)$ ;  
4 return  $M_{rd}$ ;  
5 Function  $\text{PruningInOneWay}(M, U_i, k)$   
6    $D \leftarrow \emptyset$ ;  
7   foreach  $u_i \in U_i$  do  
8      $B \leftarrow \{(u_1, u_2) \in M : u_1 = u_i \vee u_2 = u_i\}$ ;  
9     if  $|B| \leq k$  then continue; /* no need to prune */  
10    foreach  $(u_1, u_2) \in B$  do  
11      if  $\min\_rank_i(u_1, u_2) \geq k$  then  
12        /*  $(u'_1, u'_2)$  cannot be pruned here */  
13         $B \leftarrow \{(u'_1, u'_2) \in B : \mathbf{s}(u_1, u_2) \not\leq \mathbf{s}(u'_1, u'_2)\}$ ;  
14     $D \leftarrow D \cup B$ ;  
15 return  $D$ ;
```

matches is denoted by M_{rd} , where each entity is involved in nearly k candidate matches, due to the weak ordering of partial order.

Algorithm 1 first partitions entity match set M into each block B where all pairs contain the same entity (Line 8). Then, it checks each entity pair $(u_1, u_2) \in B$, and prunes entity pairs such that $\min_rank \geq k$ (Lines 10–12). Finally, the retained pairs in B are added into the output match set.

Algorithm 1 first takes $O(|M_c||M_{at}|)$ time to pre-compute the similarity vectors. When processing U_i ($i = 1, 2$), the pruning step (Lines 7–13) checks at most $|M_c|$ pairs, and each time it spends $O(3|U_{3-i}||M_{at}|)$ time to compute \min_rank_i , prune pairs in B and store the retained pairs in D . So, the overall time complexity of Algorithm 1 is $O(|M_c||M_{at}|(|U_1| + |U_2|))$. In practice, similarity vector construction is the most time-consuming part, while the pruning step only needs to check a small amount of entities in U_1 or U_2 .

V. RELATIONAL MATCH PROPAGATION

Given an ER graph $\mathcal{G} = (V, E, l_v, l_e)$ and an entity match $u_1 \simeq u_2$ in it, the relational match propagation infers how likely each unresolved entity pair $p \in V$ is a match based on the structure of \mathcal{G} , i.e. $\Pr[m_p | u_1 \simeq u_2]$. In this section, we first consider a basic case that unresolved entity pairs are neighbors of a match in \mathcal{G} . Then, we generalize it to the case that unresolved pairs are reachable from several matches. In the basic case, we resolve entity pairs between two value sets of a relationship pair, and define the consistency between relationships to measure the portion of values containing matched counterparts in another value set. The consistency and the prior match probabilities of entity pairs are further combined to obtain “tight” posterior match probabilities. In the general case, we propose a Markov model on paths from matches to unresolved ones to find the match probability bounds.

A. Consistency Between Relationships

Functional/inverse functional properties are ideal for match propagation. For example, *wasBornIn* is a functional property,

and the born places of two persons in a match must be identical. However, we cannot just rely on functional/inverse functional properties, since many relationships are multi-valued and only a part of the values may match. Thus, we define the consistency between relationships as follows.

Let r_1 and r_2 be two relationships in two KBs. We assume that, given the condition that $u_1 \simeq u_2 \wedge u'_1 \in N_{u_1}^{r_1}$, the probability of the event $\exists u'_2 : (u'_2 \in N_{u_2}^{r_2} \wedge u'_1 \simeq u'_2)$ is subject to a binary distribution with parameter ϵ_1 . Symmetrically, we define parameter ϵ_2 . We use ϵ_1 and ϵ_2 to represent the consistency for two relationships r_1 and r_2 , respectively:

$$\begin{aligned} \epsilon_1 &= \Pr[\exists u'_2 : u'_2 \in N_{u_2}^{r_2} \wedge u'_1 \simeq u'_2 | u_1 \simeq u_2, u'_1 \in N_{u_1}^{r_1}], \\ \epsilon_2 &= \Pr[\exists u'_1 : u_1 \in N_{u_1}^{r_1} \wedge u'_2 \simeq u'_1 | u_2 \simeq u_1, u'_2 \in N_{u_2}^{r_2}]. \end{aligned} \quad (3)$$

where $N_{u_1}^{r_1}, N_{u_2}^{r_2}$ are the value sets of relationships r_1, r_2 w.r.t. entities u_1, u_2 , respectively.

To estimate ϵ_1 and ϵ_2 , we use the value distribution on the initial entity matches M_{in} . For an entity pair $(u_1, u_2) \in M_{in}$, we introduce a latent random variable $L_{u_1, u_2}^{r_1, r_2} = |M_{u_1, u_2}^{r_1, r_2}|$, where $M_{u_1, u_2}^{r_1, r_2}$ denotes the set of entity matches in $N_{u_1}^{r_1} \times N_{u_2}^{r_2}$. Note that we omit r_1, r_2 in $L_{u_1, u_2}^{r_1, r_2}$ and $M_{u_1, u_2}^{r_1, r_2}$ to simplify notations. Similar to [39], we make an assumption on the entity sets: no duplicate entities exist in each entity set. Hence, L_{u_1, u_2} is also the number of entities in $N_{u_1}^{r_1}$ (or $N_{u_2}^{r_2}$) which appear in M_{u_1, u_2} . Based on the latent variable L_{u_1, u_2} , the likelihood probability of $(N_{u_1}^{r_1}, N_{u_2}^{r_2}, L_{u_1, u_2})$ is

$$\Pr[N_{u_1}^{r_1}, N_{u_2}^{r_2}, L_{u_1, u_2}] = \prod_{i=1,2} \binom{|N_{u_i}^{r_i}|}{L_{u_1, u_2}} \left(\frac{\epsilon_i}{1-\epsilon_i}\right)^{L_{u_1, u_2}} (1-\epsilon_i)^{|N_{u_i}^{r_i}|}. \quad (4)$$

Then, we use the maximum likelihood estimation to obtain ϵ_1 and ϵ_2 :

$$\max_{\epsilon_1, \epsilon_2, L_{\cdot, \cdot}} \prod_{(u_1, u_2) \in M_{in}} \Pr[N_{u_1}^{r_1}, N_{u_2}^{r_2}, L_{u_1, u_2}]. \quad (5)$$

Since each L_{u_1, u_2} is an integer variable, the brute-force optimization can cost exponential time. Next, we present an optimization process. Let $\zeta = \frac{\epsilon_1 \epsilon_2}{(1-\epsilon_1)(1-\epsilon_2)}$ and $\xi(\epsilon_1, \epsilon_2) = (1-\epsilon_1)^{b_1} (1-\epsilon_2)^{b_2}$, where $b_1 = \sum |N_{u_1}^{r_1}|, b_2 = \sum |N_{u_2}^{r_2}|$. We simplify (5) to $\max_{\epsilon_1, \epsilon_2} \xi(\epsilon_1, \epsilon_2) \prod \max_{L_{u_1, u_2}} c_{L_{u_1, u_2}} \zeta^{L_{u_1, u_2}}$, where $c_{L_{u_1, u_2}} = \binom{|N_{u_1}^{r_1}|}{L_{u_1, u_2}} \binom{|N_{u_2}^{r_2}|}{L_{u_1, u_2}}$. Notice that $c_i \zeta^i = c_j \zeta^j$ has only one solution for different integers i, j . Thus, the curves $c_{L_{u_1, u_2}} \zeta^{L_{u_1, u_2}}$ ($0 \leq L_{u_1, u_2} \leq L_M$) can have at most $\binom{L_M+1}{2}$ common points, where $L_M = \min\{|N_{u_1}^{r_1}|, |N_{u_2}^{r_2}|\}$. Therefore, $\max_{L_{u_1, u_2}} c_{L_{u_1, u_2}} \zeta^{L_{u_1, u_2}}$ is an $O(L_M^2)$ -piecewise continuous function, and the product of these $O(L_M^2)$ -piecewise continuous functions is an $O(\max\{|N_{u_1}^{r_1}|^4, |N_{u_2}^{r_2}|^4\})$ -piecewise continuous function. As a result, we can optimize (5) by solving $O(\max\{|N_{u_1}^{r_1}|^4, |N_{u_2}^{r_2}|^4\})$ continuous optimization problems with two variables, which runs efficiently.

B. Match Propagation to Neighbors

A basic case is that the unresolved entity pairs are adjacent to a match $u_1 \simeq u_2$ in \mathcal{G} . We consider the neighbors with the same edge label, i.e. relationship pair (r_1, r_2) , together. Then, our goal is to identify matches between $N_{u_1}^{r_1}$ and $N_{u_2}^{r_2}$.

Let $M_{u_1, u_2} \subseteq N_{u_1}^{r_1} \times N_{u_2}^{r_2}$ denote a set of entity matches. We consider two factors about how likely M_{u_1, u_2} can be the correct match result of $N_{u_1}^{r_1} \times N_{u_2}^{r_2}$: (1) the prior match probabilities of matches without neighborhood information; (2) the consistency of the relationships. The match probability of M_{u_1, u_2} given $u_1 \simeq u_2$ is:

$$\Pr[M_{u_1, u_2} | u_1 \simeq u_2] = \frac{1}{Z} f(M_{u_1, u_2} | N_{u_1}^{r_1}, N_{u_2}^{r_2}) \times g(M_{u_1, u_2} | N_{u_1}^{r_1}) g(M_{u_1, u_2} | N_{u_2}^{r_2}), \quad (6)$$

where Z is the normalization factor. $f(M_{u_1, u_2} | N_{u_1}^{r_1}, N_{u_2}^{r_2})$ is the prior match probability. $g(M_{u_1, u_2} | N_{u_1}^{r_1}), g(M_{u_1, u_2} | N_{u_2}^{r_2})$ are the consistency of M_{u_1, u_2} w.r.t. $N_{u_1}^{r_1}, N_{u_2}^{r_2}$, respectively.

Without considering neighborhood information, the prior match probability $f(M_{u_1, u_2} | N_{u_1}^{r_1}, N_{u_2}^{r_2})$ is defined as the likelihood function of M_{u_1, u_2} :

$$f(M_{u_1, u_2} | N_{u_1}^{r_1}, N_{u_2}^{r_2}) = \prod_{p \in M_{u_1, u_2}} \Pr[m_p] \times \prod_{p \in N_{u_1}^{r_1} \times N_{u_2}^{r_2} \setminus M_{u_1, u_2}} (1 - \Pr[m_p]), \quad (7)$$

where $\Pr[m_p]$ denotes the prior probability of entity pair p being a match, and $1 - \Pr[m_p]$ denotes the prior probability of p being a non-match.

Let $\pi_1(M_{u_1, u_2}) = \{u'_1 | (u'_1, u'_2) \in M_{u_1, u_2}\}$. Note that when u_1 and u_2 form a match, each entity $u'_1 \in \pi_1(M_{u_1, u_2})$ is a neighbor of u_1 for relationship r_1 such that $\exists u'_2 : u'_2 \in N_{u_2}^{r_2} \wedge u'_2 \simeq u'_1$. Based on ϵ_1 , the consistency of M_{u_1, u_2} given $N_{u_1}^{r_1}$ is defined as follows:

$$g(M_{u_1, u_2} | N_{u_1}^{r_1}) = \epsilon_1^{|\pi_1(M_{u_1, u_2})|} (1 - \epsilon_1)^{|N_{u_1}^{r_1}| - |\pi_1(M_{u_1, u_2})|}. \quad (8)$$

$\pi_2(M_{u_1, u_2})$ and $g(M_{u_1, u_2} | N_{u_2}^{r_2})$ can be defined similarly.

Finally, we obtain the posterior match probability of $u'_1 \simeq u'_2$ by marginalizing $\Pr[u'_1 \simeq u'_2, M_{u_1, u_2} | u_1 \simeq u_2]$:

$$\Pr[u'_1 \simeq u'_2 | u_1 \simeq u_2] = \sum_{M_{u_1, u_2}} \Pr[u'_1 \simeq u'_2, M_{u_1, u_2} | u_1 \simeq u_2] = \sum_{M_{u_1, u_2}: (u'_1, u'_2) \in M_{u_1, u_2}} \Pr[M_{u_1, u_2} | u_1 \simeq u_2], \quad (9)$$

where M_{u_1, u_2} is selected over $(N_{u_1}^{r_1} \times N_{u_2}^{r_2}) \cap V$.

Example. Let $(u_1, u_2) = (y:Tim, d:Tim)$, r_1 and r_2 denote the relationship *directed*, $\epsilon_1 = \epsilon_2 = 0.9$, and $\Pr[m_p] \equiv 0.5$ (implying all pairs are viewed as the same). From Figure 1, we can find that $N_{u_1}^{r_1} = \{y:Cradle, y:Player\}$ and $N_{u_2}^{r_2} = \{d:Cradle, d:Player\}$. Thus, when $M_{u_1, u_2} = \{(y:Cradle, d:Cradle), (y:Player, d:Player)\}$, $\Pr[M_{u_1, u_2} | u_1 \simeq u_2] = 0.5^3 \times 0.95^4 \approx 0.1$; when $M'_{u_1, u_2} = \{(y:Cradle, d:Player)\}$, $\Pr[M'_{u_1, u_2} | u_1 \simeq u_2] = 0.5^3 \times 0.95^2 \times 0.05^2 \approx 0.0003$. So, M_{u_1, u_2} is more likely to be the match set within $N_{u_1}^{r_1} \times N_{u_2}^{r_2}$. Furthermore, $\Pr[y:Cradle \simeq d:Cradle] \approx 0.99$, whereas $\Pr[y:Cradle \simeq d:Player] \approx 0.01$.

C. Distant Match Propagation

The above match propagation to neighbors only estimates the match probabilities of direct neighbors of an entity match, which lacks the capability of discovering entity matches far away. In the following, we extend it to a more general case,

called *distant match propagation*, where a match reaches an unresolved entity pair through a path.

Intuitively, given a match (u_1, u_2) and an unresolved pair (u'_1, u'_2) , the distant propagation process can be modeled as a path consisting of the entity pairs from (u_1, u_2) to (u'_1, u'_2) , where each unresolved pair can be inferred as a match via its precedent. Assume that there is a path $(u_1^0, u_2^0), (u_1^1, u_2^1), \dots, (u_1^l, u_2^l)$ in \mathcal{G} , where $(u_1^0, u_2^0) = (u_1, u_2)$ and $(u_1^l, u_2^l) = (u'_1, u'_2)$. According to the chain rule of conditional probability, we have

$$\begin{aligned} & \Pr[u'_1 \simeq u'_2 | u_1^0 \simeq u_2^0] \\ & \geq \Pr[u'_1 \simeq u'_2, u_1^1 \simeq u_2^1, \dots, u_1^l \simeq u_2^l | u_1^0 \simeq u_2^0] \\ & = \prod_{i=1}^l \Pr[u_1^i \simeq u_2^i | u_1^0 \simeq u_2^0, \dots, u_1^{i-1} \simeq u_2^{i-1}] \quad (10) \\ & = \prod_{i=1}^l \Pr[u_1^i \simeq u_2^i | u_1^{i-1} \simeq u_2^{i-1}], \end{aligned}$$

where the last “=” holds because we assume that this propagation path satisfies the Markov property [22]. Inequation (10) gives a lower bound for $\Pr[u'_1 \simeq u'_2 | u_1^0 \simeq u_2^0]$. The largest lower bound is selected to estimate $\Pr[u'_1 \simeq u'_2 | u_1 \simeq u_2]$. We estimate $\Pr[u'_1 \simeq u'_2 | u_1 \simeq u_2]$ in Algorithm 2.

VI. MULTIPLE QUESTIONS SELECTION

Based on the relational match propagation, unresolved entity pairs can be inferred by human-labeled matches. However, different questions have different inference capabilities. In this section, we first describe the definition of inferred match set and the multiple questions selection problem. Then, we design a graph-based algorithm to determine the inferred match set for each question. Finally, we formulate the benefit of multiple questions and design a greedy algorithm to select the best questions.

A. Question Benefits

We follow the so-called *pairwise question interface* [5], [6], [7], [12], [14], [17], where each question is whether an entity pair is a match or not. Let Q be a set of pairwise questions. Labeling Q can be defined as a binary function $H : Q \rightarrow \{0, 1\}$, where for each question $q \in Q$, $H(q) = 1$ means that q is labeled as a match, while $H(q) = 0$ indicates that q is labeled as a non-match.

Given the labels H , we propagate the labeled matches in H to unresolved pairs. The set of entity pairs that can be inferred as matches by H is

$$\text{inferred}(H) = \bigcup_{q \in Q: H(q)=1} \text{inferred}(q), \quad (11)$$

$$\text{inferred}(q) = \{p \in C : \Pr[m_p | m_q] \geq \tau\}, \quad (12)$$

where C is the unresolved entity pairs and τ is the precision threshold for inferring high-quality matches. We evaluate $\text{inferred}(q)$ in Section VI-B.

Since non-matches are quadratically more than matches in the ER problem [1], the labels to the ideal questions should infer as many matches as possible. Thus, we define the benefit

function of Q as the expected number of matches can be inferred by labels to Q , which is

$$\text{benefit}(Q) = \mathbb{E}[|\text{inferred}(H)| | Q]. \quad (13)$$

The ER algorithm can ask each question with the greatest benefit iteratively; however, there is a latency caused by waiting for workers to finish the question. Assigning multiple questions to workers simultaneously in one human-machine loop is a straightforward optimization to reduce the latency. Since workers in crowdsourcing platforms are paid based on the number of solved questions, the number of questions should be smaller than a given budget. Thus, the *optimal multiple questions selection* problem is to

$$\begin{aligned} & \text{maximize} && \text{benefit}(Q), \\ & \text{s.t.} && Q \subseteq C, \quad |Q| \leq \mu, \end{aligned} \quad (14)$$

where μ is the constraint on the number of questions asked.

B. Discovery of Inferred Match Set

In order to obtain the benefit for each question set Q , we need to compute $\text{inferred}(q)$ for each $q \in Q$. To estimate $\Pr[m_p | m_q]$ in $\text{inferred}(q)$, we define the length of a directed edge (v, v') in probabilistic ER graph \mathcal{F} as $\text{length}(v, v') = -\log f(v, v') = -\log \Pr[m_{v'} | m_v]$. According to the definition of $\Pr[m_p | m_q]$, $\Pr[m_p | m_q] = e^{\text{dist}(q, p)}$, where $\text{dist}(q, p)$ is the distance of the shortest path from q to p . As a result, the condition $\Pr[m_p | m_q] \geq \tau$ can be interpreted as $\text{dist}(q, p) \leq \zeta = -\log \tau$. Note that edge (v, v') can be removed when $\Pr[m_{v'} | m_v] = 0$ to avoid $\log 0$.

The all-pairs shortest path algorithms can efficiently compute $\text{inferred}(q)$ for every q . Since most $|\text{inferred}(q)|$ should be smaller than $|C|$, we choose to apply binary trees rather than an array of size $|C|$ to maintain distances. We depict our modified Floyd-Warshall algorithm in Algorithm 2. In Lines 1–2, for every q , we create a binary tree $\text{bt}(q)$ to store the inferred pairs as well as their corresponding lengths, and a binary tree $\text{bt}^{-1}(q)$ to store pairs inferring q as well as their corresponding lengths. In Lines 3–5, the edge whose length is not greater than ζ would be stored into binary trees. In Lines 6–11, we modify the dynamic programming process in the original FloydWarshall algorithm. Since the number of pairs which can be inferred is significantly less than $|C|$, the inner loop in Lines 9–11 iterate only over the set of distances which are likely to be updated. Lines 13–14 extract the inferred match sets from binary trees.

Since each binary tree contains at most $|C|$ elements, $|R| \leq |\text{bt}(q).\text{val}| \leq |C|$. The loop in Lines 6–11 takes $O(|C|^3)$ time in total. The time complexity of Algorithm 2 is $O(|C|^3)$.

C. Multiple Questions Selection

Since the match propagation works independently for each label, the event that an entity pair p is inferred as a match by labels H is equivalent to the event that p is inferred by $q \in Q$ such that $H(q) = 1$. When H is not labeled, p is resolved as a match if and only if at least one question that can resolve

Algorithm 2: DP-based inferred match set discovery

Input: Probabilistic ER graph \mathcal{F} , candidate question set C , distance threshold ζ
Output: Set B of inferred match sets for all questions

```

1 foreach  $q \in C$  do
2    $\lfloor$  Initialize two empty binary trees  $\text{bt}(q), \text{bt}^{-1}(q)$ ;
3 foreach  $(q, p) \in C \times C$  do
4   if  $\text{length}(q, p) \leq \zeta$  then
5      $\lfloor$   $\text{bt}(q)[p] \leftarrow \text{length}(q, p); \text{bt}^{-1}(p)[q] \leftarrow \text{length}(p, q)$ ;
6 foreach  $q \in C$  do
7   foreach  $p \in \text{bt}(q).\text{val}$  do
8      $R \leftarrow \{r \in \text{bt}^{-1}(q).\text{val} : \text{bt}(q)[p] + \text{bt}^{-1}(q)[r] \leq \zeta\}$ ;
9     foreach  $r \in R$  do
10       $d \leftarrow \text{bt}(q)[p] + \text{bt}^{-1}(q)[r]$ ;
11       $\lfloor$   $\text{bt}(q)[r] \leftarrow d; \text{bt}^{-1}(r)[p] \leftarrow d$ ;
12  $B \leftarrow \emptyset$ ;
13 foreach  $q \in C$  do
14    $\lfloor$   $\text{inferred}(q) \leftarrow \text{bt}(q).\text{val}; B \leftarrow B \cup \{\text{inferred}(q)\}$ ;
15 return  $B$ ;
```

p as a match is labeled as a match. Given the question set Q , the probability that p can be resolved as a match by labels is

$$\Pr[p \in \text{inferred}(H) | Q] = 1 - \prod_{q \in Q: p \in \text{inferred}(q)} (1 - \Pr[m_q]), \quad (15)$$

where $\text{inferred}(H)$ is defined in Eq. (11), representing the matches that can be inferred after Q is labeled by workers.

The benefit of question set Q is formulated as the expected size of the inferred matches by labels H :

$$\begin{aligned} \text{benefit}(Q) &= \mathbb{E}[|\text{inferred}(H)| | Q] \\ &= \sum_{p \in C} \Pr[p \in \text{inferred}(H) | Q]. \end{aligned} \quad (16)$$

Now, we want to select a set of questions that can maximize the benefit. We first prove the hardness of the multiple questions selection problem. Then, we describe a greedy algorithm to solve it.

Theorem 1. *The problem of optimal multiple questions selection is NP-hard.*

Proof. The optimization version of the set cover problem is NP-hard. Given an element set $U = \{1, 2, \dots, n\}$ and a collection S of sets whose union equals U , the set cover problem aims to find the minimum number of sets in S whose union also equals U . This problem can be reduced to our multiple questions selection problem in polynomial time. Assume that the vertex set of an ER graph is $\{p_1, p_2, \dots, p_n\} \cup \{p_s | s \in S\}$, the edge set is $\{(p_s, p_k) : k = 1, 2, \dots, n \wedge s \in S \wedge k \in s\}$, all the prior match probabilities are 1, the precision threshold is 1, $\Pr[p_k | p_s] = 1$ and $\Pr[p_s | p_k] = 0$, for all $k = 1, 2, \dots, n, s \in S$ satisfying $k \in s$. Because the benefit is equal to the number of covered elements in U , the optimal solution of the multiple questions selection problem is also that of the set cover problem. Thus, the multiple questions selection problem is NP-hard. \square

Theorem 2. *$\text{benefit}(Q)$ is an increasing submodular function.*

Algorithm 3: Greedy multiple questions selection

Input: Probabilistic ER graph \mathcal{F} , candidate question set C , precision threshold τ , question number μ
Output: Selected question set Q

- 1 $Q \leftarrow \emptyset$; $PQ \leftarrow \{(q, \text{benefit}(\{q\})) \mid q \in C\}$;
- 2 **while** $|Q| < \mu$ **do**
- 3 $q, \Delta q \leftarrow PQ.\text{pop}()$; $q', \Delta q' \leftarrow PQ.\text{top}()$;
- 4 **while** $\Delta q > 0$ **do**
- 5 $\Delta q \leftarrow \text{benefit}(Q \cup \{q\}) - \text{benefit}(Q)$;
- 6 **if** $\Delta q \geq \Delta q'$ **then** $Q \leftarrow Q \cup \{q\}$; **break**;
- 7 **else** $PQ.\text{push}((q, \Delta q))$;
- 8 $q, \Delta q \leftarrow PQ.\text{pop}()$; $q', \Delta q' \leftarrow PQ.\text{top}()$;
- 9 **if** $\Delta q \leq 0$ **then break**;
- 10 **return** Q ;

Proof. Let $b_p(Q)$ represent $\Pr[p \in \text{inferred}(H) \mid Q]$. For every $p \in C$ and two disjoint subsets $Q, Q' \subseteq C$, we have

$$b_p(Q \cup Q') = b_p(Q) + b_p(Q') - b_p(Q)b_p(Q').$$

Thus, $b_p(Q \cup Q') - b_p(Q) = b_p(Q')(1 - b_p(Q)) \geq 0$. Since $\text{benefit}(Q) = \sum_{p \in C} b_p(Q)$, it is an increasing function.

Also, for every $p \in C, Q \subseteq C$ and $q_1, q_2 \in C \setminus Q$ such that $q_1 \neq q_2$, we have

$$\begin{aligned} b_p(Q \cup \{q_1, q_2\}) + b_p(Q) - b_p(Q \cup \{q_1\}) - b_p(Q \cup \{q_2\}) \\ = b_p(\{q_2\})(b_p(Q) - b_p(Q \cup \{q_1\})) \leq 0. \end{aligned}$$

Thus, $b_p(Q \cup \{q_1\}) + b_p(Q \cup \{q_2\}) \geq b_p(Q \cup \{q_1, q_2\}) + b_p(Q)$. Since $\text{benefit}(Q) = \sum_{p \in C} b_p(Q)$, it is a submodular function. Together, we prove that $\text{benefit}(\cdot)$ is an increasing submodular function. \square

Since Eq. (16) is monotonic and submodular, the multiple questions selection problem can be solved by using submodular optimization. We design Algorithm 3, which gives a $(1 - \frac{1}{e})$ -approximation guarantee. This algorithm selects questions greedily with the highest gain in benefits (i.e. Δq). We also leverage the lazy evaluation of the submodular function to improve the efficiency [40]. Specifically, we maintain a priority queue PQ over each candidate question q ordered by the gain in benefits Δq in descending order. Based on the submodular property, when the gain in benefits Δq of the picked question q is greater than that of the top question q' in PQ , q is the question with the largest gain in benefits. We use an array to store $b_p(Q)$, such that Δq can be obtained in $O(|C|)$ time. The overall time complexity of Algorithm 3 is $O(\mu|C|^2)$, where μ is the number of questions asked in each loop and C is the set of unresolved entity pairs in the ER graph.

VII. TRUTH INFERENCE

After the questions are labeled by workers, we design an error-tolerant model to infer truths (i.e. matches and non-matches) from the imperfect labeling, which facilitates updating the (probabilistic) ER graph and resolving isolated entities.

A. Error-Tolerant Inference

As the labels completed by the workers on crowdsourcing platforms may contain errors, we assign one question to multiple workers and use their labels to infer the posterior match probabilities. We leverage the worker probability model [41], which uses a single real number to denote a worker w 's quality $\lambda^w \in (0, 1]$, i.e. the probability that w can correctly label a question. Since crowdsourcing platforms, e.g., Amazon MTurk¹ offers a qualification test for their workers, we reuse a worker's precision in this test as her quality. The posterior probability of question q being a match is

$$\begin{aligned} & \Pr[m_q \mid W_T, W_F] \\ &= \frac{\Pr[m_q] \Pr[W_T, W_F \mid m_q]}{\Pr[m_q] \Pr[W_T, W_F \mid m_q] + \Pr[\bar{m}_q] \Pr[W_T, W_F \mid \bar{m}_q]} \\ &= \frac{\Pr[m_q]}{\Pr[m_q] + \Pr[\bar{m}_q] \prod_{w \in W_T} \frac{1 - \lambda^w}{\lambda^w} \prod_{w \in W_F} \frac{\lambda^w}{1 - \lambda^w}}, \end{aligned} \quad (17)$$

where W_T denotes the set of workers labeling q as a match, and W_F denotes the set of workers labeling q as a non-match.

We assign two thresholds to filter matches and non-matches based on consistent labels. Entity pairs with a high posterior probability (e.g., ≥ 0.8) are regarded as matches, while pairs with a low posterior probability (e.g., ≤ 0.2) are non-matches. Others are considered as inconsistent and remain unresolved. One possible reason for the inconsistency is that these questions are too hard. For a hard question q , we set $\Pr[m_q]$ to $\Pr[m_q \mid W_T, W_F]$ for reducing its benefit, thereby it is less possible to be asked more times. Next, we infer matches based on the consistent labels and re-estimate the probability of each edge in \mathcal{F} using new matches and non-matches.

B. Inference for Isolated Entity Pairs

As an exception, there may exist a small amount of isolated entity pairs which do not occur in any relationship triples. In this case, the match propagation cannot infer their truths, and the question selection algorithm has to ask these pairs one by one. To avoid such an inefficient polling, we reuse the similarity vectors and the partial order relations obtained in Section IV to train a classifier for these isolated pairs.

Given an isolated entity pair p , let A_p denote the set of its attribute matches. We define the set N_p of retained matches with similar attributes to p by $N_p = \{p' \in M_{rd} : \text{Jaccard}(A_p, A_{p'}) \geq \psi\}$, where Jaccard calculates the similarity between two sets of attribute matches. ψ is a threshold, and we set $\psi = 0.9$ for high precision. Since we only allow matches to propagate in the ER graph, most obtained labels are matches. Therefore, we treat all unresolved pairs in N_p as non-matches to balance the proportions of different labels.

Next, we use N_p and the labels as training data, and scikit-learn² to train a random forest classifier with default parameter to predict whether p is a match. The random forest finds the unresolved pairs in N_p whose similarity vectors are close to known matches.

¹<https://www.mturk.com/>

²<https://scikit-learn.org>

TABLE II: Statistics of the datasets

	#Entities	#Attributes	#Relationships	#Matches
IIMB	365 / 365	12 / 12	15 / 15	365
D-A	2.61K / 64.3K	3 / 3	1 / 1	5.35K
I-Y	15.1M / 3.04M	14 / 36	15 / 33	77K
D-Y	3.12M / 3.04M	684 / 36	688 / 33	1.31M

TABLE III: F1-score and number of questions with real workers

	Remp		HIKE		POWER		Corleone	
	F1	#Q	F1	#Q	F1	#Q	F1	#Q
IIMB	95.3%	10	84.4%	70	82.4%	70	94.7%	173
D-A	97.7%	60	93.3%	80	94.8%	70	94.5%	161
I-Y	70.9%	110	68.1%	270	69.3%	240	64.5%	402
D-Y	87.2%	130	86.4%	500	84.3%	500	76.3%	1166

VIII. EXPERIMENTS AND RESULTS

In this section, we conduct a thorough evaluation on the effectiveness of our approach Remp, by comparing with state-of-the-art methods followed by an in-depth investigation on each part of Remp (as outlined in Section III-B).

Datasets. We use one benchmark dataset and three real-world datasets widely used in previous work [12], [16], [28], [29]. Table II lists their statistics.

- IIMB is a small, synthetic benchmark dataset in OAEI³ containing two KBs with identical attributes and relationships.
- DBLP-ACM (abbr. D-A)⁴ is a dataset about publications and authors. The original version uses a text field to store all authors of a publication. Here, we split it and create authorship triples. In the case that an author has multiple representations on the original dataset, we follow [42] to extend the gold standard with author matches.
- IMDB-YAGO (abbr. I-Y) is a large dataset about movies and actors. Following [29], we generate the gold standard based on “external links” in Wikipedia pages.
- DBpedia-YAGO (abbr. D-Y) is a large dataset with heterogeneous attributes and relationships. We use the same version as in [12], [28].

Competitors. We compare Remp with three state-of-the-art crowdsourced ER approaches, namely, HIKE [12], POWER [16] and Corleone [9]. We have introduced them in Section II. Since POWER and Corleone are designed for tabular data, we follow HIKE to partition entities into different clusters and deploy POWER and Corleone on each entity cluster. Specifically, IIMB, D-A and I-Y have clear type information, which is directly used to partition entities; for D-Y which does not have clear type information, we reuse the partitioning algorithm presented in HIKE.

Setup. We implement Remp and all competing methods (as their codes are not available) in Python 3 and C++, and strictly follow each competitor’s reported parameters in the respective paper. All our codes are open sourced⁵. All experiments are conducted on a workstation with an Intel Xeon 3.3GHz CPU and 128GB RAM. For Remp, we uniformly assign $k = 4$,

³http://islab.di.unimi.it/content/im_oaei/2019/

⁴https://dbs.uni-leipzig.de/en/research/projects/object_matching

⁵<https://github.com/nju-websoft/Remp>

$\tau = 0.9$ and $\mu = 10$, and use 0.3 as the label similarity threshold. Similar to [9], [12], [16], we first prune out all definite non-matches (outlined in Section IV), and all methods take the same retained entity matches M_{rd} as input.

A. Remp vs. State of the Art

We set up two experiments, one is with real workers and the other is with simulated workers. The evaluation metrics are the F1-score and the number of questions (#Q).

Experiment with real workers. We publish the questions selected by each approach on Amazon MTurk. Each question is labeled by five workers to decide whether the two entities refer to the same object in the real world. We leverage the common worker qualifications to avoid spammers, i.e. we only allow workers with an approval rate of at least 95%. Furthermore, we reuse the label to each question for all approaches. Thus, all approaches can receive the same label to the same question. In total, 651 real workers labeled 3,484 questions.

The results are presented in Table III, and we have the following findings – (1) Remp consistently achieves the best F1-score with the fewest questions. (2) Remp improves the F1-score moderately, and reduces the number of questions significantly. (3) Specifically, compared with the second best result, Remp reduces the average number of questions by 85.7%, 14.3%, 54.2% and 74.0% on IIMB, D-A, I-Y and D-Y, respectively. To summarize, Remp achieves the best F1-score and saves the number of questions, especially when the dataset contains various relationships (e.g., D-Y).

Experiment with simulated workers. We also generate simulated workers who give wrong labels to questions with a fixed probability (called *error rate*). We follow HIKE to set the error rate of simulated workers to 0.05, 0.15 and 0.25.

Figure 3 shows the comparison results and we make several observations – (1) All approaches obtain stable F1-scores, indicating their robustness in handling imperfect labeling. (2) Remp consistently obtains the highest F1-score, and beats the second best result by 0.4%, 3.0%, 1.6%, 8.0% on IIMB, D-A, I-Y and D-Y, respectively. This is attributed to Remp’s robustness in uncovering matches with low literal similarities as compared to its competitors. For example, literal information is insufficient on I-Y and D-Y, thereby causing errors in the partial order of HIKE and POWER as well as the rules of Corleone. (3) Remp needs considerably fewer questions on IIMB, I-Y and D-Y. Compared with the second best result, Remp reduces the average number of questions by 79.9%, 26.7%, 62.5% and 71.4% on IIMB, D-A, I-Y and D-Y, respectively. One reason is that there are many types of entities on these datasets, and most matches are linked by relationships of different domain/range types. However, HIKE, POWER and Corleone cannot infer these matches efficiently. (4) On the D-A dataset, Remp only reduces six more questions than POWER, because in the ER graph there are many isolated components but only one type of relationship, making Remp have to check them all.

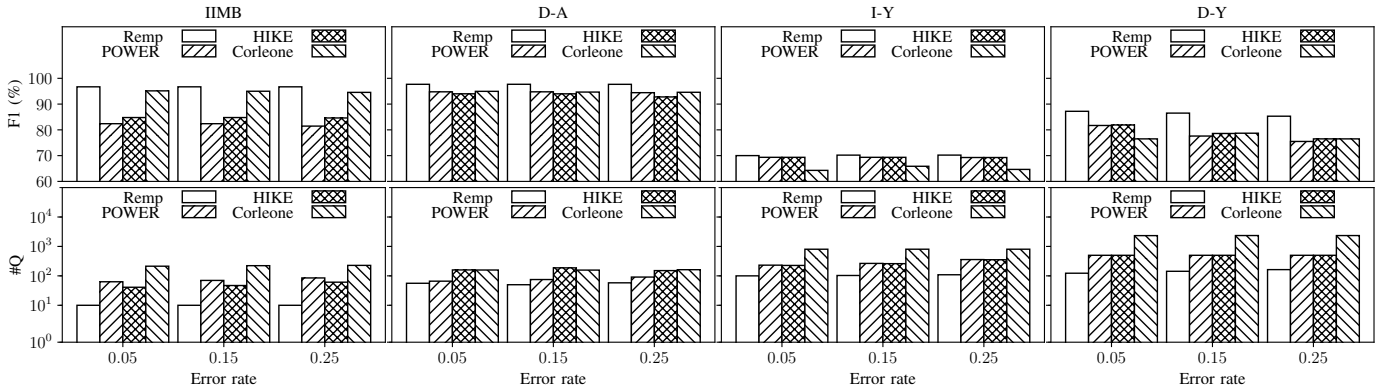


Fig. 3: F1-score and number of questions w.r.t. simulated workers of varying error rates

TABLE IV: Effectiveness of attribute matching

	#Ref. matches	Remp			Remp w/o 1:1 matching		
		Precision	Recall	F1	Precision	Recall	F1
I-Y	4	100%	100%	100%	40.0%	100%	57.1%
D-Y	19	90.9%	52.6%	66.7%	52.4%	57.9%	55.0%

B. Internal Evaluation of Remp

In this section, we evaluate how each major module of Remp contributes to its overall performance.

Effectiveness of attribute matching. For the I-Y dataset, we reuse the gold standard created by SiGMA [29]. For the D-Y dataset, we follow the recommendation of YAGO and extract 19 attribute matches from the subPropertyOf links⁶ as the gold standard. Note that it is not necessary to match attributes for the other two datasets. We employ the conventional precision, recall and F1-score as our evaluation metrics.

As depicted in Table IV, Remp performs perfectly on the I-Y dataset but gains a relatively low recall on the D-Y dataset, and the 1:1 matching constraint helps Remp improve the precision. We observe that Remp fails to identify several attribute matches when the attribute pairs rarely appear in M_{in} (i.e. entity pairs from exact string matching), or when the values are dramatically different (e.g., the *icd10* value for *dbp:Trigeminal_neuralgia* is “G44.847”, but for *yago:Trigeminal_neuralgia* is “G-50.0”). We argue that our attribute matches are sufficient to ER, since the first type of missing matches only helps resolve a small portion of entities but increases the running time of building similarity vectors, while the second type requires extra value processing/correction steps before computing the similarities.

Effectiveness of partial order based pruning. To test the performance of the entity pair pruning module in Remp, we employ two metrics: (i) the reduction ratio (RR), which is the proportion of pruned candidates, and (ii) the pair completeness (PC), which is the proportion of true matches preserved in candidate/retained matches. We also use the error rate of optimal monotone classifier defined in [15] to measure the incorrectness of the partial order.

As shown in Table V, candidate matches contain most true matches on IIMB, D-A and I-Y, but only 88.7% of

TABLE V: Effectiveness of partial order based pruning

$k = 4$	Candidate matches		Retained matches			
	#Pairs	PC	#Pairs (RR)	PC	#Edges	Error rate
IIMB	593	97.8%	516 (13.0%)	97.8%	1K	1.91%
D-A	24.2K	97.9%	12.4K (49.0%)	97.7%	7.6K	0.37%
I-Y	2.44B	98.0%	3.86M (99.6%)	97.4%	0.16M	0.65%
D-Y	2.70B	88.7%	13.1M (99.7%)	84.8%	5.34M	1.64%

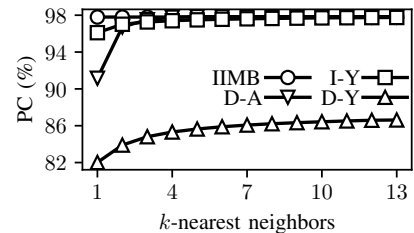


Fig. 4: Pair completeness w.r.t. k -nearest neighbors

true matches on the D-Y dataset. This is because on this dataset 8.4% of the entities in the true matches lack labels. On IIMB and D-A, Remp has a relatively low RR, because the true matches account for 61.6% and 22.1% of the candidate matches, respectively. On I-Y and D-Y, the PC of retained matches is close to that of candidate matches, but most candidate matches are pruned. This indicates that the entity pair pruning module is effective. We notice that the error rate on each dataset is nearly perfect, but the other monotonicity-based approaches (i.e., POWER and HIKE) achieve worse accuracy (see Table III). The main reason is that our partial order is restricted to neighbors of each entity pair, where errors do not propagate to the whole candidate match set.

Furthermore, the pair completeness of retained matches w.r.t. varying k is shown in Figure 4. The pair completeness converges quickly on IIMB, D-A and I-Y but slowly on D-Y, because many matches have only one or two shared attributes, making the partial order work inefficiently.

Effectiveness of match propagation. We additionally compare the match propagation module of Remp with two collective, non-crowdsourcing ER approaches: PARIS [28] and SiGMA [29]. More details about them have been given in Section II. To assess the real propagation capability of Remp, we ignore the classifier for handling isolated entity pairs. We randomly sample different portions of entity matches as the

⁶http://webdam.inria.fr/paris/yd_relations.zip

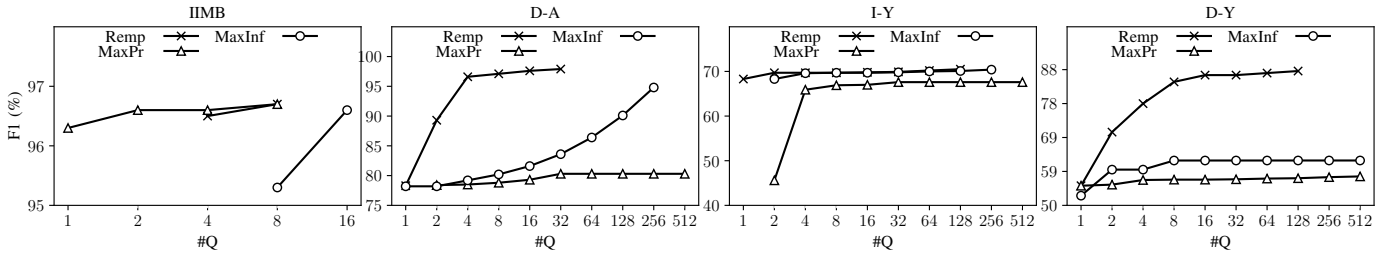


Fig. 5: F1-score of Remp, MaxInf and MaxPr w.r.t. varying numbers of questions

TABLE VI: F1-score w.r.t. varying portions of seed matches

		% of matches			
		20	40	60	80
IIMB	Remp	97.5%	98.6%	99.7%	99.7%
	PARIS	96.0%	96.5%	97.0%	97.4%
	SiGMa	97.6%	98.6%	99.0%	99.6%
D-A	Remp	93.3%	97.2%	98.9%	99.7%
	PARIS	71.3%	79.1%	86.2%	92.5%
	SiGMa	92.7%	94.9%	96.7%	98.4%
I-Y	Remp	41.2%	63.4%	78.8%	90.6%
	PARIS	34.8%	57.9%	75.4%	89.0%
	SiGMa	34.0%	58.5%	76.1%	89.3%
D-Y	Remp	83.2%	91.4%	95.0%	99.7%
	PARIS	82.2%	84.7%	87.2%	89.5%
	SiGMa	33.6%	57.4%	75.3%	89.1%

seeds for Remp, PARIS and SiGMa. The experiments are repeated five times and the F1-score is reported in Table VI. We observe that Remp achieves the best F1-score on D-A, I-Y and D-Y. On IIMB (20% of matches), the F1-score of Remp is slightly worse than that of SiGMa, because SiGMa can obtain matches between isolated entities based on their literal similarities directly. Overall, Remp can achieve the highest F1-score in most cases.

Effectiveness of question selection benefit. We implement two alternative heuristics as baselines, namely MaxInf and MaxPr, to evaluate the question selection benefit. We set $\mu = 1$ and use ground truths as labels. MaxInf selects the questions with the maximal inference power. MaxPr chooses the questions with the maximal match probability. Figure 5 depicts the result and each curve starts when the F1-score is greater than 0. We find (1) Remp always achieves the best F1-score with much less number of questions. (2) MaxPr obtains the lowest F1-score except on the IIMB dataset, because it does not consider how many matches can be inferred by the new question. (3) MaxInf performs worse than Remp, as it often chooses non-matches as the questions, making it find fewer matches than Remp using the same number of questions. This experiment demonstrates that our *benefit* function is the most effective one.

Effectiveness of multiple questions selection. Table VII depicts the F1-score, the number of questions (#Q) and the number of loops (#L) of the multiple questions selection module (with ground truth as labels), in term of different question number thresholds per round ($\mu = 1, 5, 10, 20$), and our findings are as follows – (1) Remp achieves a stable F1-score on all datasets. (2) The number of questions increases when μ increases, especially when $\mu = 10, 20$. This is probably

TABLE VII: F1-score and number of questions with different question number thresholds per round

	$\mu = 1$		$\mu = 5$		$\mu = 10$		$\mu = 20$		
	F1	#Q	#L	F1	#Q	#L	F1	#Q	#L
IIMB	96.7%	8	8	96.7%	10	2	96.7%	20	2
D-A	97.8%	52	52	97.8%	60	12	97.7%	60	6
I-Y	71.4%	102	102	71.3%	105	21	71.3%	110	11
D-Y	87.3%	127	127	87.2%	135	27	87.3%	140	14

TABLE VIII: F1-score of inference on isolated entity pairs

	Isolated matches	Remp	Random forest
IIMB	0.3%	95.3%	0.0%
D-A	0.4%	97.7%	13.7%
I-Y	28.1%	70.9%	66.3%
D-Y	60.4%	87.2%	84.5%

because Remp always asks μ questions in one human-machine loop, and it has to ask an extra batch of questions when some questions with large *benefit* are labeled as non-matches. Although asking multiple questions in one loop increases the monetary cost, it reduces 75%–94.1% number of loops when $\mu = 20$.

Effectiveness of inference on isolated entity pairs. We examine the performance of the random forest classifier in each dataset in the experiments with real workers. As depicted in Table VIII, the classifier achieves poor performance on IIMB and D-A. Due to the tiny proportion of isolated entity pairs in these two datasets, this is probably caused by occasionality. When the portions of isolated matches increase on I-Y and D-Y, the classifier achieves comparable performance to Remp. This demonstrates that Remp can infer enough matches for resolving the entire dataset even if the ER graph does not cover all candidate matches.

Efficiency Analysis. We run each algorithm three times to record the running time on each of the four datasets. The average running time of Algorithm 1 on four datasets is 1s, 8s, 3.9h and 3.6h, the average running time of Algorithm 2 is 0.476s, 6.7s, 109s and 1.07h, and the average running time of Algorithm 3 is 0.128s, 1.27s, 78.5s and 1.25h. We follow the analysis in [10] to evaluate the performance of Remp on 25%, 50%, 75% and 100% of candidate (retained) entity matches M_c (M_{rd}) on the D-Y dataset. As depicted in Figure 6, the running time of Algorithm 1 and Algorithm 2 increase linearly as the number of entity pairs increases. The running time of Algorithm 3 on 25% and 50% of retained entity matches are close. This is probably because the sizes of some inferred match sets do not increase significantly.

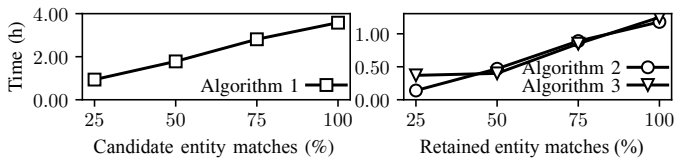


Fig. 6: Running time w.r.t. different portion of entity pairs

IX. CONCLUSION

In this paper, we proposed a crowdsourced approach leveraging relationships to resolve entities in KBs collectively. Our main contributions are a partial order based pruning algorithm, a relational match propagation model, a constrained multiple questions selection algorithm and an error-tolerant truth inference model. Compared with existing work, our experimental results demonstrated superior ER accuracy and much less number of questions. In future work, we plan to combine transitive relation, partial order and match propagation together as a hybrid ER approach.

ACKNOWLEDGMENTS

This work was partially supported by the National Key R&D Program of China under Grant 2018YFB1004300, the National Natural Science Foundation of China under Grants 61872172, 61772264 and 91646204, and the ARC under Grants DP200102611 and DP180102050. Zhifeng Bao is the recipient of Google Faculty Award.

REFERENCES

- [1] L. Getoor and A. Machanavajjhala, "Entity resolution: Tutorial," http://users.umiaccs.umd.edu/~getoor/Tutorials/ER_VLDB2012.pdf, 2012.
- [2] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, "Duplicate record detection: A survey," *IEEE TKDE*, vol. 19, no. 1, pp. 1–16, 2007.
- [3] K. Sun, Y. Zhu, and J. Song, "Progress and challenges on entity alignment of geographic knowledge bases," *International Journal of Geo-Information*, vol. 8, no. 2, pp. 77–101, 2019.
- [4] J. Bleiholder and F. Naumann, "Data fusion," *ACM Computing Surveys*, vol. 41, no. 1, pp. 1–41, 2009.
- [5] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng, "Leveraging transitive relations for crowdsourced joins," in *SIGMOD*. ACM, 2013, pp. 229–240.
- [6] S. E. Whang, P. Lofgren, and H. Garcia-Molina, "Question selection for crowd entity resolution," *Proc. of the VLDB Endowment*, vol. 6, no. 6, pp. 349–360, 2013.
- [7] D. Firmani, B. Saha, and D. Srivastava, "Online entity resolution using an oracle," *Proc. of the VLDB Endowment*, vol. 9, no. 5, pp. 384–395, 2016.
- [8] A. Arasu, M. Götz, and R. Kaushik, "On active learning of record matching packages," in *SIGMOD*. ACM, 2010, pp. 783–794.
- [9] C. Gokhale, S. Das, A. Doan, J. F. Naughton, N. Rampalli, J. Shavlik, and X. Zhu, "Corleone: Hands-off crowdsourcing for entity matching," in *SIGMOD*. ACM, 2014, pp. 601–612.
- [10] S. Das, P. S. GC, A. Doan, J. F. Naughton, G. Krishnan, R. Deep, E. Arcaute, V. Raghavendra, and Y. Park, "Falcon: Scaling up hands-off crowdsourced entity matching to build cloud services," in *SIGMOD*. ACM, 2017, pp. 1431–1446.
- [11] K. Qian, L. Popa, and P. Sen, "Active learning for large-scale entity resolution," in *CIKM*. ACM, 2017, pp. 1379–1388.
- [12] Y. Zhuang, G. Li, Z. Zhong, and J. Feng, "Hike: A hybrid human-machine method for entity alignment in large-scale knowledge bases," in *CIKM*. ACM, 2017, pp. 1917–1926.
- [13] P. F. Patel-Schneider, P. Hayes, and I. Horrocks, *OWL Web ontology language semantics and abstract syntax*, W3C, 2004.
- [14] N. Vesdapunt, K. Bellare, and N. Dalvi, "Crowdsourcing algorithms for entity resolution," *Proc. of the VLDB Endowment*, vol. 7, no. 12, pp. 1071–1082, 2014.
- [15] Y. Tao, "Entity matching with active monotone classification," in *PODS*. ACM, 2018, pp. 49–62.
- [16] C. Chai, G. Li, J. Li, D. Deng, and J. Feng, "A partial-order-based framework for cost-effective crowdsourced entity resolution," *The VLDB Journal*, vol. 27, no. 6, pp. 745–770, 2018.
- [17] V. Verroios and H. Garcia-Molina, "Entity resolution with crowd errors," in *ICDE*. IEEE, 2015, pp. 219–230.
- [18] A. Marcus, E. Wu, D. Karger, S. Madden, and R. Miller, "Human-powered sorts and joins," *Proc. of the VLDB Endowment*, vol. 5, no. 1, pp. 13–24, 2011.
- [19] J. Wang, T. Kraska, M. J. Franklin, and J. Feng, "CrowdER: Crowdsourcing entity resolution," *Proc. of the VLDB Endowment*, vol. 5, no. 11, pp. 1483–1494, 2012.
- [20] V. Verroios, H. Garcia-Molina, and Y. Papakonstantinou, "Waldo: An adaptive human interface for crowd entity resolution," in *SIGMOD*. ACM, 2017, pp. 1133–1148.
- [21] S. Galhotra, D. Firmani, B. Saha, and D. Srivastava, "Robust entity resolution using random graphs," in *SIGMOD*. ACM, 2018, pp. 3–18.
- [22] V. Rastogi, N. Dalvi, and M. Garofalakis, "Large-scale collective entity matching," *Proc. of the VLDB Endowment*, vol. 4, no. 4, pp. 208–218, 2011.
- [23] C. Böhm, G. De Melo, F. Naumann, and G. Weikum, "Linda: distributed web-of-data-scale entity matching," in *CIKM*. ACM, 2012, pp. 2104–2108.
- [24] Y. Altowim, D. V. Kalashnikov, and S. Mehrotra, "Progressive approach to relational entity resolution," *Proc. of the VLDB Endowment*, vol. 7, no. 11, pp. 999–1010, 2014.
- [25] V. Efthymiou, G. Papadakis, K. Stefanidis, and V. Christophides, "Minoaner: Schema-agnostic, non-iterative, massively parallel resolution of web entities," in *EDBT*, 2019, pp. 373–384.
- [26] A. Kimmig, A. Memory, R. J. Miller, and L. Getoor, "A collective, probabilistic approach to schema mapping," in *ICDE*. IEEE, 2017, pp. 921–932.
- [27] P. Kouki, J. Pujara, C. Marcum, L. Koehly, and L. Getoor, "Collective entity resolution in familial networks," in *ICDM*. IEEE, 2017, pp. 227–236.
- [28] F. M. Suchanek, S. Abiteboul, and P. Senellart, "PARIS: Probabilistic alignment of relations, instances, and schema," *Proc. of the VLDB Endowment*, vol. 5, no. 3, pp. 157–168, 2011.
- [29] S. Lacoste-Julien, K. Palla, A. Davies, G. Kasneci, T. Graepel, and Z. Ghahramani, "SiGMa: Simple greedy matching for aligning large knowledge bases," in *KDD*. ACM, 2013, pp. 572–580.
- [30] Y. Zhuang, G. Li, Z. Zhong, and J. Feng, "PBA: Partition and blocking based alignment for large knowledge bases," in *DASFAA*. Springer, 2016, pp. 415–431.
- [31] X. Dong, A. Halevy, and J. Madhavan, "Reference reconciliation in complex information spaces," in *SIGMOD*. ACM, 2005, pp. 85–96.
- [32] T. Papenbrock, A. Heise, and F. Naumann, "Progressive duplicate detection," *IEEE TKDE*, vol. 27, no. 5, pp. 1316–1329, 2015.
- [33] J. Sun, Z. Shang, G. Li, Z. Bao, and D. Deng, "Balance-aware distributed string similarity-based query processing system," *Proc. of the VLDB Endowment*, vol. 12, no. 9, pp. 961–974, 2019.
- [34] W. Hu and C. Jia, "A bootstrapping approach to entity linkage on the semantic web," *Journal of Web Semantics*, vol. 34, pp. 1–12, 2015.
- [35] F. Naumann and M. Herschel, *An introduction to duplicate detection*. Morgan and Claypool Publishers, 2010.
- [36] M. Cheatham and P. Hitzler, "The properties of property alignment," in *ISWC Workshop on Ontology Matching*. CEUR-WS, 2014.
- [37] I. Megdiche, O. Teste, and C. Trojahn, "An extensible linear approach for holistic ontology matching," in *ISWC*, vol. LNCS 9981. Springer, 2016, pp. 393–410.
- [38] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.
- [39] D. Zhang, B. I. P. Rubinstein, and J. Gemmell, "Principled graph matching algorithms for integrating multiple data sources," *IEEE TKDE*, vol. 27, no. 10, pp. 2784–2796, 2015.
- [40] B. Mirzasoleiman, A. Badanidiyuru, A. Karbasi, J. Vondrák, and A. Krause, "Lazier than lazy greedy," in *AAAI*, 2015, pp. 1812–1818.
- [41] Y. Zheng, G. Li, Y. Li, C. Shan, and R. Cheng, "Truth inference in crowdsourcing: Is the problem solved?" *Proc. of the VLDB Endowment*, vol. 10, no. 5, pp. 541–552, 2017.
- [42] A. Thor and E. Rahm, "MOMA - A mapping-based object matching system," in *CIDR 2007*, 2007, pp. 247–258.