Hike: A Hybrid Human-Machine Method for Entity Alignment in Large-Scale Knowledge Bases

Yan Zhuang, Guoliang Li, Zhuojian Zhong, Jianhua Feng
Department of Computer Science, Tsinghua University, Beijing China
zhuang-y14@mails.tsinghua.edu.cn, liguoliang@tsinghua.edu.cn, zhuojiaz@uci.edu

ABSTRACT

With the vigorous development of the World Wide Web, many large-scale knowledge bases (*KBs*) have been generated. To improve the coverage of *KBs*, an important task is to integrate the heterogeneous *KBs*. Several automatic alignment methods have been proposed which achieve considerable success. However, due to the inconsistency and uncertainty of large-scale *KBs*, automatic techniques for *KBs* alignment achieve low quality (especially recall). Thanks to the open crowdsourcing platforms, we can harness the crowd to improve the alignment quality. To achieve this goal, in this paper we propose a novel hybrid human-machine framework for large-scale *KB* integration. We first partition the entities of different *KBs* into many smaller blocks based on their relations. We then construct a partial order on these partitions and develop an inference model which crowdsources a set of tasks to the crowd and infers the answers of other tasks based on the crowdsourced tasks. Next we formulate the question selection problem, which, given a monetary budget $\mathcal{B}$, selects $\mathcal{B}$ crowdsourced tasks to maximize the number of inferred tasks. We prove that this problem is NP-hard and propose greedy algorithms to address this problem with an approximation ratio of $1 - 1/e$. Our experiments on real-world datasets indicate that our method improves the quality and outperforms state-of-the-art approaches.

KEYWORDS

Entity Alignment; Crowdsourcing; Knowledge base

1 INTRODUCTION

Knowledge base has become one of the most important research directions in the World Wide Web, which provides a uniform framework for data sharing and interpreting. A growing number of large-scale knowledge bases (*KBs*) have been created (e.g., *DBPedia* [18], *YAGO* [34]), and many relevant applications (e.g., question answering [11], machine reading [27], knowledge support [2] and semantic search [1]) have also been constructed.

However, different *KBs* are heterogeneous and inconsistent in nature, and they can complement each other. For example, *DBPedia* has many entities but few classes while *YAGO* has hundreds of thousands of classes. Linking *DBPedia* with *YAGO* enables us to provide more semantic information for *DBPedia* which can facilitate many applications. Knowledge bases alignment is proposed to address this problem, which aims to integrate different *KBs* to improve the coverage and quality of *KBs*. There are three main elements in *KBs*: *class*, *relation* (or *property*) and *entity* (*instance*). Knowledge base alignment aims to link them together for disambiguation. Entity alignment is more challenging and important because there are large numbers of entities and it can also benefit class and relation alignment. In this paper we focus on entity alignment which links the entities from different knowledge bases that refer to the same real-world entities, e.g., US and American, Peking and Beijing. Although there are many automatic algorithms to align entities [14, 17, 24, 33], they usually attain low recall (e.g., 70%), due to the inconsistency and noise in the data. Fortunately, with the development of crowdsourcing techniques, we can harness the crowd to improve the alignment quality [23] [9]. Specifically, we ask the crowd to label entity pairs, by checking whether two entities in a pair refer to the same entity.

There are several challenges to utilize the crowd to align *KBs*. Firstly, the knowledge bases have a large number of entities and it is expensive to label every entity pair. More importantly, the crowd is not free and we need to pay each worker for labeling a pair. Thus we are usually given a monetary budget which can be easily converted to the constraint of the number of crowdsourced pairs, and an important problem is to select $\mathcal{B}$ entity pairs to crowdsource and deduce the answer of other pairs based on the answers of the selected pairs. Secondly, the crowd may return incorrect answers and it is important to tolerate the errors.

To address these challenges, in this work we propose a hybrid human-machine framework. To cope with large-scale alignment, we first partition the whole *KBs* into many smaller blocks such that the entities in the same block are highly probable to be aligned while the entities in different blocks are less likely to be matched. Then we only need to align the entity pairs in the same block. To reduce monetary costs, we propose a partial order based approach, which defines a partial order on the entity pairs. That is, if an entity pair is labeled to be aligned, then the pairs preceding this pair (i.e., has higher probability) based on the partial order are also taken as aligned. In this way, we can deduce the results for a large set of entity pairs. We prove that the problem of "*selecting $\mathcal{B}$ crowdsourced pairs that maximizes the number of deduced pairs*" is NP-hard, and we propose two greedy algorithms to solve the problem. We also propose an effective method to tolerate the crowd errors. To summarize, we make the following contributions.

- We present a hybrid human-machine framework which harnesses the crowd to align entities in large-scale *KBs* (Section 3).
- We propose a partition algorithm to partition the entities which can reduce the alignment scale significantly (Section 4).

CIKM'17 , November 6–10, 2017, Singapore, Singapore
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-4918-5/17/11…$15.00
https://doi.org/10.1145/3132847.3132912

• We define a partial order on the entity pairs and propose a partial-order-based method (Section 5). We prove the question selection problem as NP-hard and further propose two effective algorithms to select the most beneficial questions (Section 6) in an error-tolerant manner (Section 7).
• We have conducted extensive experiments on real *KBs* to evaluate our methods. Experimental results show that our method outperforms state-of-the-art approaches by 8% in terms of recall (Section 8).

The rest of this paper is organized as follows. In Section 2, we introduce the preliminaries and summarize related works. Section 3 shows an overview of our framework. The *KB* partition strategy is presented in Section 4, and the partial order is defined in Section 5. We present the question selection algorithms in Section 6 and discuss the error-tolerant techniques in Section 7. Experimental results are reported in Section 8 and we conclude in Section 9.

2 PRELIMINARIES

2.1 Problem Definition

The Resource Description Framework (RDF[1]) is a language for representing information about resources in the World Wide Web. Here we follow and simplify the *RDF* model to define our knowledge base. In our model, a *KB* contains *entities* (or *instances*), *classes*, *literals*, *relations* and *properties*. Resources may be divided into groups called classes (e.g. location, company). The members of a class are known as instances of the class (e.g. Beijing, Apple Inc.). A *literal* is a string, date or number. A *relation* defines the relationship between two instances, while a *property* captures a literal (or attribute) of an entity. A *KB* consists of multiple *facts*, where each *fact* is a *RDF* triple: ⟨*subject, predicate, object*⟩ (abbreviated as *SPO*). The *subject* here is an *entity* or *class* as defined above. The *object* is a *class*, *entity*, or *literal*, which respectively denotes (1) the category that the subject belongs to, (2) the related real-world object, or (3) an attribute that describes the subject. In this paper, we use *predicate* to refer to both property and relation. For example, the predicate *isLocatedIn* defines the relationship between Beijing and China, and another predicate *owner* defines the relationship between the Apple Inc. and Jobs.

DEFINITION 1. *(ENTITY ALIGNMENT IN KBs) Given two KBs (say K and K'), the entity alignment problem is to find the pairs of entities in K and K' that refer to the same real-world entity.*

DEFINITION 2. *(HUMAN-MACHINE ENTITY ALIGNMENT) Given a budget $\mathcal{B}$, it selects $\mathcal{B}$ questions to ask the crowd in order to maximize the number of aligned pairs, where each question contains a pair of entities and asks the crowd to identify whether the two entities refer to the same real-world one.*

2.2 Related Work

Entity Alignment on *KBs* has been studied extensively in these years (see [30] for a survey). Some existing work has investigated how to address the problem with automatic or semi-automatic approaches. SIGMA [17] is an iterative propagation algorithm which leverages both the relations and properties between entities to collectively align *KBs*. SIGMA involves experts to manually select the related relationships and properties and thus is a semi-automatic method. VMI [25] directly uses the vector space model to generate multiple vectors for entities, and builds a set of inverted indexes to get the primary matching candidates. However, the matching properties also need to be specified by users. These semi-automatic approaches only ask experts to assist their automatic algorithms and do not focus on hybrid human-machine approaches. These methods have two key differences from ours. Firstly, they cannot utilize the feedback of experts to deduce the results of other pairs. Secondly, they take the experts as oracles which do not consider the fact that the crowd may make errors. PARIS [33] is an automatic algorithm which provides a holistic probabilistic solution to align large-scale *KBs*. It computes alignments not only for entities, but also for classes and relations. PBA [42] improves the efficiency and scalability of PARIS, by using multiple partition techniques and local collective approaches. They are the state-of-the-art automatic methods but they have a low recall. Our hybrid framework takes PARIS and PBA as baselines and focuses on improving the alignment quality with crowdsourcing.

There are also many studies on leveraging crowdsourcing to address the problem. CROWDMAP [32] addressed the Ontology alignment problem via microtask crowdsourcing, while it made no considerations on how to improve efficiency which is crucial for large-scale dataset. Wang et al. [37] and Vesdapunt et al. [35] studied how to utilize the transitivity to reduce the number of questions. Wang et al. proposed a heuristic approach which first computed a similarity value for each candidate pair and then presented the candidate pairs to the crowd in a decreasing order to maximize the deducing power. Based on this work, Vesdapunt et al. proposed another heuristic approach and provided a better worst-case performance guarantee. Obviously the transitivity can reduce the cost, but it may introduce some negative effects on quality and latency. ACD [38] and GCER [39] are another two crowdsourcing ER frameworks which mostly focus on matching quality. They achieve high quality at high monetary budget. CROWDER [36] and POWER [5] adopt a two-step framework, in which they first generate a candidate set of matching pairs by automatic methods, and then introduce crowdsourcing to check the matching results. However, these crowd-based entity resolution methods can only deal with relatively small structure data. Although Power uses a partial-order approach, it cannot support *KBs* that have complex large-scale structures. Thus our techniques (e.g., partial order and question selection) that can be applied to large-scale *KBs* are different from Power. There are also some works address the entity alignment problem by optimizing the choice of an action with respect to the expected information gain [15][10]. Yet most of them focus on the alignment quality. We design crowd-based entity alignment algorithms to align two large-scale *KBs* and focus on both the monetary cost and the alignment quality.

There are also studies on error tolerances [7, 19–22, 40, 41].

3 THE HIKE FRAMEWORK

We present the HIKE framework in Figure 1. It takes two *KBs* as input, and generates matching entity pairs as output. Firstly, these *KBs* are fed into automatic machine-based algorithms to divide all the entity pairs into two parts: the matched pairs and unmatched pairs. (1) For matched pairs, we aim to find the false positives and

[1]http://www.w3.org/TR/rdf-primer/

add them into unmatched pairs, with the target of improving the metric precision. (2) For the unmatched pairs, similarly our target is to find the matched pairs, which improves the metric recall. The techniques of the two parts are similar in nature. For illustration purpose, we take the part that processes unmatched pairs as an example, and discuss the main components below.

Step 1: Entity Partition. The real-world *KBs* have tens of millions of entities and it is rather expensive to enumerate every pair of entities. Many automatic alignment algorithms leverage partition-based algorithms to reduce the matching scale. They usually utilize the class hierarchy to partition the *KBs*. In Hike, we leverage the predicate, which is a very important hint to partition the entities, since intuitively, (1) similar entities should have the similar predicates, e.g., persons may have *BirthPlace* and *BirthDate* as properties in many *KBs*; (2) the same predicates imply that the corresponding entities should be similar, e.g., the entities having predicate *BirthDate* should be person. Based on the intuitions, we propose a predicate-based entity partition method, which partitions the entities with similar predicates into the same block, which can prune a large number of entities in different blocks (Section 4). Then we can compose and select questions based on entities in the same block, and ask the crowd for answers.

Step 2: Partial Order Construction. Our idea is to select a set of questions such that based on their answers, we can maximally induce the answers of other (unasked) questions. Based on this, we define a partial order in questions. Intuitively, for a pair of entities p_1=$\langle e_1, e_2\rangle$, if they indeed refer to the same real-world entity (e_1=e_2), then for another pair of entities p_2=$\langle e_3, e_4\rangle$, if the following holds: (1) p_2 *shares very similar predicates with* p_1; (2) *the similarity of each predicate in* p_2 *is higher than the corresponding predicate in* p_1, we can infer that the pairs in p_2 (i.e., e_3 and e_4) are to be the same one with high confidence. The partial order can also be applied the other way around. That is, if an entity pair with a set of predicates refer to different objects, then another pair with smaller similarity in each of the corresponding predicates can be inferred as different entities. Based on the partial order, we find that different entity pairs have different inference power, and pairs with larger inference power should be asked first (Section 5).

Step 3: Question Selection. Based on the partial order, we aim to select a set of representative pairs that can maximize the number of induced questions. The basic idea is that each entity pair in the partial order has its inference power and the pair with largest inference power should be asked first. However, the ground truth of each question is unknown, and we model the expected inference power of the selected entity pairs. Thus the pairs with the maximum inference power are selected to assign (Section 6).

Step 4: Error Tolerance. Since the crowd may return incorrect results, we need to tolerate errors. More importantly, if the partial order on some entity pairs is incorrect, it will propagate the inference errors. We propose worker-quality control and error-propagation deduction techniques to tolerate these errors (Section 7). Our technique is general, and most recent advanced inference techniques can also be applied in our framework.

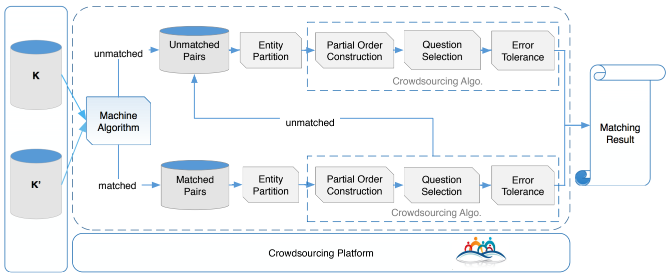


Figure 1: Overview of Hike framework.

4 PREDICATE-BASED ENTITY PARTITION

In order to partition the two *KBs* based on predicates, we define the similarity on predicates. For ease of presentation, we first introduce some notations.

Consider two *KBs* K and K'. Let $\Phi = \{p_1, p_2, \cdots, p_m\}$ and $\Phi' = \{p'_1, p'_2, \cdots, p'_n\}$ denote the corresponding predicate sets. Let $T(p_i) = \{(s,o)|(s,p_i,o) \in K, p_i \in \Phi\}$ denote the set of tuples with predicate p_i, where a tuple contains a subject s and object o such that (s, p_i, o) is a triple in K.

Definition 3. *(Predicate Similarity) The similarity between two predicates* p_i *and* p'_j *is defined as*

$$\textsc{sim}(p_i, p'_j) = \frac{|T(p_i) \cap T'(p'_j)|}{|T(p_i) \cup T'(p'_j)|}, \tag{1}$$

where $(s,o) \in T(p_i) \cap T'(p'_j)$ if (s, p_i, o) is a triple in K and (s, p'_j, o) is a triple in K', and $|\cdot|$ is the size of a set.

For each predicate $p_i \in K$, we find its most similar predicate $p'_j \in K'$ such that $\textsc{sim}(p_i, p'_j) \geq \textsc{sim}(p_i, p'_x)$. Similarly, for each predicate $p'_i \in K'$, we find its most similar predicate $p_j \in K$ such that $\textsc{sim}(p_j, p'_i) \geq \textsc{sim}(p_x, p'_i)$. We call these predicate pairs *matching predicate pairs.* Formally, $\mathbb{P} = \{(p_i \in K, p'_j \in K')|\textsc{sim}(p_i, p'_j) \geq \textsc{sim}(p_i, p'_x) \text{ for } x \neq j\} \cup \{(p_j \in K, p'_i \in K')|\textsc{sim}(p_j, p'_i) \geq \textsc{sim}(p_j, p'_x) \text{ for } x \neq i\}$ denotes the set of matching predicate pairs. We then associate the entity pairs to the corresponding matching predicate pairs. For each matching predicate pair $pp^k = (p_i, p'_j) \in \mathbb{P}$, we compute two sets of entities $S(pp^k) = \{s|(s, p_i, o) \in K\}$ and $S'(pp^k) = \{s'|(s', p'_j, o') \in K'\}$, where the former is the set of subjects with predicate p_i and the latter is the set of subjects with predicate p'_j. For example, in Figure 2, $S(pp^1) = \{e^1_1, e^2_1\}$ and $S'(pp^1) = \{e^1_2, e^2_2\}$, and $S(pp^2) = \{e^1_1, e^2_1, e^3_1\}$, $S'(pp^2) = \{e^1_2, e^2_2, e^3_2\}$.

Next we discuss how to partition entity pairs into different blocks. Obviously, if two matching predicates share many common entities, we should partition them together. Based on this observation, we define the similarity between matching predicate pairs.

Definition 4. *(Similarity of Matching Predicate Pairs) Given two matching predicate pairs* pp^i *and* pp^j*, the similarity* ρ *is computed as below:*

$$\rho(pp^i, pp^j) = \frac{\cos(S(pp^i), S(pp^j)) + \cos(S'(pp^i), S'(pp^j))}{2}. \tag{2}$$

where cos is the cosine similarity between two sets. Obviously, this similarity is normalized, e.g. $\rho(pp^i, pp^j) \in [0,1]$ and symmetric, e.g. $\rho(pp^i, pp^j) = \rho(pp^j, pp^i)$.

Algorithm 1: ESPS: ENTITY SET PAIRS SELECTION

```
    Input: $KB_1$, $KB_2$ : Two knowledge bases for matching; $\tau$ :
           Similarity Threshold;
    Output: $\mathcal{P}$ : Set of partitioned entity pairs
1   begin
2       W = GenerateSimMatrix(KB1,KB2);
3       PP = GeneratePredPairs(W);
4       Q = GenerateQueue(PP, W);
5       $\delta$ = 1;
6       while $\delta \geq \tau$ do
7           Q = MergePred(Q,W,$\delta$);
8           W = CalcPartitionSim(Q,W);
9           $\delta = \delta - 1/10$;
10          if Q do not change then
11              Break;
12      $\mathcal{P}$ = GeneratePartitionPairs(Q);
```

Given the definition of cluster similarity, the *KBs* partition algorithm can be presented as a modified hierarchical agglomerative clustering(HAC) algorithm, which is widely applied in the IR field. The entity pair sets can be merged together iteratively in descending order of the similarity to build the hierarchy. Traditional *HAC* algorithm [28] is often criticized on its lower efficiency, so we modified the merging process to improve the rate of convergence. Furthermore, some of the largest *KBs* usually have only few thousands predicates, the modified HAC algorithm can fit the clustering scale well. The detail of the algorithm can be divided into three parts which is demonstrated below.

Initialization: The algorithm accepts the matched instance pairs produced by the machine algorithm from two *KBs* as prior alignment data and generates an $m \times n$ similarity matrix $\boldsymbol{W}$ of the predicates (line 2). Then, $min(m, n)$ predicate pairs are generated according to the similarity matrix. Each predicate pair here have an entity set pair (line 3). After that a priority queue is initialized to cache the predicate partitions in descending order of the number of the entity set pairs (line 4). This constructs the bottom level of the hierarchy. Each node represents its own partition which produces $min(m, n)$ partitions in the priority queue. The cut level of similarity is initialized with 1 (line 5), and the gap between the levels of hierarchy is set to 0.1.

Hierarchical clustering: This is the main stage of the algorithm and is designed to iteratively cluster the partition hierarchy. While the cut level of similarity is more than a specified similarity threshold (line 6), the algorithm iteratively merges the partitions at a certain level from the priority queue according to the threshold (line 7), and recalculate the similarity matrix $\boldsymbol{W}$ (line 8). The entity set pairs belong to the merged predicate pairs are also taken a union accordingly. The cut level of similarity is decreased by 0.1 in each iteration (line 9). The process will terminate until either the cut level of similarity is less than the threshold or there is no possibility to merge more partitions (lines 6~11).

Entity partition generation: The final stage generates the set of the partitioned entity set pairs according to the merging result of the priority queue (line 13).

Formally, we partition $\mathbb{P}$ into several partitions $\mathbb{P}_1, \mathbb{P}_2, \cdots, \mathbb{P}_z$. Each partition is a subset of $\mathbb{P}$ and the union is exactly $\mathbb{P}$. For each partition $\mathbb{P}_i$, we generate its set of entities that follow a predicate in

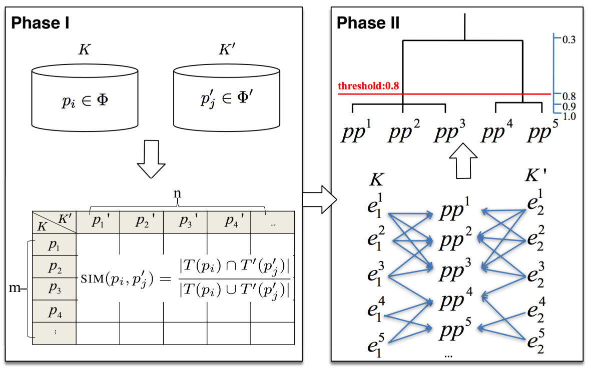

Phase I
K
K'
$p_i \in \Phi$
$p'_j \in \Phi'$
n
m
K'
K
p_1'
p_2'
p_3'
p_4'
p_1
p_2
p_3
p_4
$\text{SIM}(p_i, p'_j) = \frac{|T(p_i) \cap T'(p'_j)|}{|T(p_i) \cup T'(p'_j)|}$
Phase II
threshold:0.8
0.3
0.8
0.9
1.0
pp^1 pp^2 pp^3 pp^4 pp^5
K
K'
e_1^1 e_1^2 e_1^3 e_1^4 e_1^5
e_2^1 e_2^2 e_2^3 e_2^4 e_2^5

Figure 2: Predicate-based entity partition. (Phase I is to produce the predicate pairs, and phase II is to partition the *KBs* by clustering the predicate pairs.)

$\mathbb{P}$, i.e., $P_i = \{(s, s')|(s, p_i, o) \in K, (s', p'_j, o') \in K', (p_i, p'_j) \in \mathbb{P}_i\}$. In this way, we partition the knowledge bases to several entity blocks $P_1, P_2, \cdots$, and P_z.

EXAMPLE 1. *Take Fig.2 as an example. After phase I, the predicate pairs $pp^1 \sim pp^5$ are generated from the similarity matrix. Then, the set of entities on each pp^k can be calculated as $S(pp^1) = \{e_1^1, e_1^2\}$, $S'(pp^1) = \{e_2^1, e_2^2\}$, $S(pp^2) = \{e_1^1, e_1^2, e_1^3\}$, $S'(pp^2) = \{e_2^1, e_2^2, e_2^3\}$, $S(pp^3) = \{e_1^1, e_1^2, e_1^3\}$, $S'(pp^3) = \{e_2^1, e_2^2, e_2^3\}$, $S(pp^4) = \{e_1^3, e_1^4, e_1^5\}$, $S'(pp^4) = \{e_2^3, e_2^4, e_2^5\}$, $S(pp^5) = \{e_1^4, e_1^5\}$, and $S'(pp^5) = \{e_2^4, e_2^5\}$. As mentioned above, the cluster similarity of pp^1, pp^2 is $\rho(pp^1, pp^2) = 0.82$. We can also get $\rho(pp^2, pp^3) = 1$, $\rho(pp^4, pp^5) = 0.82$, $\rho(pp^2, pp^4) = 0.33$ and $\rho(pp^2, pp^5) = 0$ in the same way. Given threshold of 0.8, we can cluster $\{pp^1, pp^2, pp^3\}$ as one partitioned predicate pairs and $\{pp^4, pp^5\}$ as another partitioned predicate pairs after two rounds of iterations, and the final partitioned entity pairs $\{(e_1^i, e_2^j)|i, j = 1, 2, 3\}$ and $\{(e_1^i, e_2^j)|i, j = 3, 4, 5\}$ are generated.*

The complexity of the naive HAC algorithm is $O(n^3)$ because it enumerates every pair of pp^k and calculates their similarity in each of $n - 1$ iterations. Obviously this algorithm is expensive. We specify a gap between levels and cluster all the pp^k whose cluster similarity are less than specified threshold in the gap. By this way, the convergence is speeded up. Furthermore, we divide the similarity into no more than 10 gaps which makes the number of iteration a constant. Therefore the complexity of the algorithm is reduce to $O(n^2)$. Taking into account the scale of the data (usually no more than one thousand), this algorithm is applicable.

5 PARTIAL ORDER

As mentioned in Section 3, partial order can be used to maximally induce the answers of unasked questions. Thus for each partition P, we define a partial order on the entity pairs in the partition and utilize the partial order to align entities..

Partial Order Set. Let $P = \{(e_i, e_j)|e_i \in K, e_j \in K'\}$ denote a set of entity pairs in a partition. For simplicity, each entity pair $(e_i, e_j) \in P$ is denoted by p_{ij}. We use pp^k to denote the k-th predicate pair of P, and s_{ij}^k to denote the similarity of p_{ij} on predicate pp^k. (How to compute s_{ij}^k will be discussed later.) We define a partial order set $(P, \prec)$ in a partition as an entity set P with a partial order $\prec$.

Definition 5 (Partial Order). *The partial order $\prec$ can be formally defined as follows: Given two pairs $p_{ij} = (e_i, e_j), p_{i'j'} = (e_{i'}, e_{j'})$. For each pp^k, $s_{ij}^k \geq s_{i'j'}^k$, and $\sum_k(s_{ij}^k) > \sum_k(s_{i'j'}^k)$, then $p_{ij} \succ p_{i'j'}$, we say p_{ij} and $p_{i'j'}$ are comparable and p_{ij} precedes $p_{i'j'}$ or $p_{i'j'}$ succeeds p_{ij}.*

Similarity Computation. The predicate pair pp^k can be divided into three types: the name or label pair of p_{ij}, the property pair of p_{ij}, and relation pair of p_{ij}. Different type of predicates should be treated differently when we compute similarity s_{ij}^k. Here we use vector space mode [31] and Cosine similarity to compute the s_{ij}^k. For name or label pair, we build a name vector for each entity in p_{ij} which consists of terms segmented from the entity's name or label. For each property pair, we build a property vector for each entity in p_{ij} which consists of terms segmented from the entity's literal property. For each relation pair, we build a relation vector for each entity in p_{ij} which consists of the related entity's name or literal properties and do not propagate along the relations. Then, s_{ij}^k can be computed by the Cosine similarity of the correspondence vectors of p_{ij}.

The total entity pair similarity can be defined as the weighted summation of s_{ij}^k which is formally described as:

$$s_{ij} = \sum_{k=1}^{m} \omega_k \cdot s_{ij}^k \tag{3}$$

where ω_k is the weight of different pp^k which reflects the importance of this predicate. Intuitively, the more powerful the ability of the object to determine the subject is, the more important this predicate is. In knowledge base, we use inverse functionality to denote this ability. The functionality is a quasi-function which allows one subject to map to multiple objects. For example, if everyone has different E-mails in the *KB*, the predicate *hasEmail* can be regard as a function, whereas the predicate *worksat* is not a function because one may work at different places in different periods. Although the functionality of a predicate cannot determine the equivalency of the objects, it provides a good evidence. Deviating from [13], we define the functionality of an predicate as below:

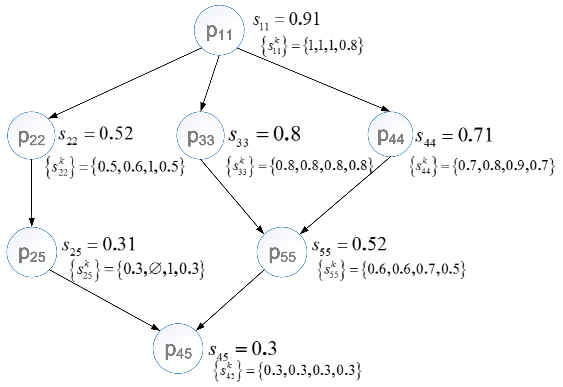
Definition 6. *(Functionality) Given a KB K and a predicate $p_i \in \Phi$, we denote the SPO triples in K on p_i as $(s, p_i, o) \in K$. The functionality of predicate p_i is ratio of the number of distinct subjects to the total number of triples with predicate p_i, i.e.,*

$$qf(p_i) = \frac{|\{s|\exists o : (s, p_i, o) \in K\}|}{|\{\langle s, o\rangle|(s, p_i, o) \in K\}|} \tag{4}$$

We define the inverse functionality as $qf^{-1}(p_i) = qf(p_i^{-1})$ where p_i^{-1} is the inverse of p_i, i.e., if there exists (s, p_i, o), then we have (o, p_i^{-1}, s). We use the inverse functionality to calculate the weights of different predicates as follows:

$$\omega_k = \frac{qf^{-1}(p_k)}{\sum_{t=1}^{m} qf^{-1}(p_t)} \tag{5}$$

Partial Order Set Generation. Because p_{ij} is generated by the Cartesian product of the two entity sets in a partition, considering the efficiency, we first prune the matching space and keep the pairs



p11
$s_{11} = 0.91$
$\{s_{11}^k\} = \{1,1,1,0.8\}$
p22
$s_{22} = 0.52$
$\{s_{22}^k\} = \{0.5,0.6,1,0.5\}$
p33
$s_{33} = 0.8$
$\{s_{33}^k\} = \{0.8,0.8,0.8,0.8\}$
p44
$s_{44} = 0.71$
$\{s_{44}^k\} = \{0.7,0.8,0.9,0.7\}$
p25
$s_{25} = 0.31$
$\{s_{25}^k\} = \{0.3,\varnothing,1,0.3\}$
p55
$s_{55} = 0.52$
$\{s_{55}^k\} = \{0.6,0.6,0.7,0.5\}$
p45
$s_{45} = 0.3$
$\{s_{45}^k\} = \{0.3,0.3,0.3,0.3\}$

Figure 3: A sketch map of partial order set.

beyond specified threshold as candidate entity pairs before generating the partial order set. We use inverted index and prefix filter [6][3] on the three type of vectors to achieve this objective. Then, given the candidate entity pairs, we can calculate the predicate similarity s_{ij} using Equation 3. To further control the matching scale, we set a threshold θ for s_{ij}. If $s_{ij} < \theta$, this entity pair will be discarded. We compare the similarity of each pp^k in the predicate pairs to build the partial order between entities in a partition, and enumerate every pair in the candidate pairs to check whether they satisfy the partial order. Thus, we construct a partial order on P. There exists many algorithms to construct a partial order, and the time complexity is $O(n^2)$ in general, but following the index-based method in [5], it will be reduce to $O(nlog^{m-1}n + |\varepsilon|)$ where m is the number of pp^k and $|\varepsilon|$ is the number of partial orders between entity pairs.

Example 2. *Suppose we have five entities in each KB whose predicate pairs are $\{\langle name, name\rangle, \langle birth\_place, born\_in\rangle, \langle birth\_date, dob\rangle, \langle article, article\rangle\}$. There produces 25 entity pairs altogether. After candidate selection and similarity threshold pruning, we get 7 pairs left. Then we construct a partial order according to their $\{s_{ij}^k\}$ as shown in Fig.3. Given the normalized weights 0.439, 0.0486, 0.0244, 0.488 for the predicate pairs of this partition, we get s_{ij} for each entity pair as labeled in the figure. The partial order and the similarity s_{ij} of the entity pair are essential in our question selection algorithm which will be demonstrated in detail in the next subsection.*

6 QUESTION SELECTION

In this section, we first introduce the inference model that is used to infer the answers from the unselected pairs, and then define the question selection problems. Lastly, we develop effective algorithms for question selection given a limited budget.

6.1 Inference Model

Question Selection. Given a partial order set $(P, \prec)$ in a block, we can infer some entity pairs' answer without asking the crowd, thus reducing the number of crowdsourced pairs. If p_{ij} is matched, the pairs (e.g., $p_{i'j'}$) preceding this entity pair in the partial order are all matched, because they have larger similarity than p_{ij}(i.e., $s_{i'j'}^k \geq s_{ij}^k$). Then these pairs can be inferred. If p_{ij} is unmatched, for the same reason, the pairs succeeding p_{ij} are all unmatched, and these pairs can also be inferred. Based on this idea, we can devise an algorithm, which asks a question and then infers the answers of other questions. Then the problem is question selection, which selects $\mathcal{B}$ questions to maximize the number of inferred pairs given $(P, \prec)$.

Offline Problem. We first consider the offline problem. That is, when we ask a question, we know whether it is a matched pair or unmatched pair. Thus for a matched pair, we can infer its preceding pairs; and for an unmatched pair, we can infer its succeeding pairs. Then we prove that the question selection problem is NP-hard by a reduction from the maximum coverage problem by setting a pair's inferred pairs as its covered elements.

Theorem 1. *The question selection problem is NP-hard.*

Proof. We prove that the question selection problem is NP-hard by a reduction from the Maximum Coverage Problem. Formally, given a universe of elements $\mathcal{U} = \{e_1, e_2, ..., e_n\}$ and a collection of subsets $S = \{S_1, S_2, ..., S_m\}$ of $\mathcal{U}$, and a number k. the Maximum Coverage Problem is to find a subset $S' \subseteq S$, s.t. $|S'| \leq k$ and the number of covered elements $| \cup_{S_i \in S'} S_i|$ is maximized. In our problem, we construct a partial order set $(P, \prec)$ and use P to correspond to the universe $\mathcal{U}$. Each pair in P has a set of inference pairs if it has been taken as the query. The sets of the inference pairs of each pair correspond to the elements of S, and sets of the inference pairs of the query set correspond to the elements of S'. To find the maximum inferred pairs corresponds to find S' with $\mathcal{B}$ elements such that the number of covered elements $|\cup_{S_i \in S'} S_i|$ is maximized. Obviously, the reduction can be done in polynomial-time, and this implies the Maximum Coverage Problem is no harder than the question selection problem. So the question selection problem is NP-hard. □

Online Problem. Next we consider the online problem - when we ask a question, we do not know whether it is a matched pair or unmatched pair. In the crowdsourcing setting, we should consider the online problem, because for each pair, we do not know whether it is matched or unmatched before we ask the crowd and we need to ask the crowd to answer each question. Note that online problem is not easier than the offline problem. Since the online problem is NP-hard, the computation becomes intractable with the increase of the number of the questions. Therefore, we design greedy algorithms to address this problem. To address this issue, we compute inference expectation of each entity pair to measure the pair' inference power, and the question selection algorithms can be developed based on the inference expectation. As each partition has distinct predicate pairs and different partitions are incomparable with each other, the question selection algorithms can be applied in all the partitions simultaneously in order to reduce the latency.

Inference Expectation. Each entity pair has inference power based on the partial order. The inference power can be defined as the number of the inferred pairs. As entity pairs can be seen as the random variables on the set 2^P, we use the mathematical expectation of the number of the inferred pairs to measure the inference power. Therefore, the problem is converted into how to select $\mathcal{B}$ entity pairs that maximize the inference expectation.

Consider a pair p_{ij}. Let $\texttt{pre}(p_{ij})$ ($\texttt{suc}(p_{ij})$) denote the set of pairs preceding (succeeding) p_{ij}. The probability that p_{ij} is matched is s_{ij} and thus if we ask p_{ij}, we have s_{ij} probability to infer its preceding pairs. Similarly, the probability that p_{ij} is unmatched is $1 - s_{ij}$ and thus if we ask p_{ij}, we have $1 - s_{ij}$ probability to infer its succeeding pairs. Based on this idea, next we define the inference expectation of an entity pair as below.

Algorithm 2: Sqs : simple question selection

```
Input: $(P, \prec)$ : Partial order set;
Output: Q: Query set
1 begin
2     for each entity pair $p_{ij} \in P$ do
3         compute $E(p_{ij})$;
4     for $x$ = 1 to $\mathcal{B}$ do
5         Select entity pair q with maximal $E(p_{ij})$ into Q;
6         Assign q to the crowds and collect the answers;
7         Refresh $E(p_{ij})$ according to the answers;
8     return Q;
```

$$E(p_{ij}) = s_{ij} \cdot |\texttt{pre}(p_{ij})| + (1 - s_{ij}) \cdot |\texttt{suc}(p_{ij})| \tag{6}$$

Given a set of questions Q, we want to compute the inference expectation of the set, i.e., $E(\texttt{Q})$. Note that we cannot simply sum up $E(p_{ij} \in \texttt{Q})$, because the inference expectation of them has overlap. For example a pair may be preceding/succeeding two question pairs in Q. To address this issue, we define an aggregated inference expectation to calculate their inference expectation.

Note that we can assume that the pairs in Q have no preceding/succeeding relationships, because if two pairs have such relationships, we can remove one of them as one can be inferred by another. Given a pair p'_{ij}, next we discuss how to compute the probability that p'_{ij} can be inferred based on questions in Q, i.e., $E(p'_{ij}|\texttt{Q})$. There are three cases shown as below.

• **Case 1:** p'_{ij} precedes Q (i.e., p'_{ij} precedes one of the questions in Q. Note that p'_{ij} cannot precede a question and succeed a question because the questions in Q have no preceding/succeeding relationships). Let $\texttt{Q}^\texttt{p}$ denote the set of questions in Q that are preceded by p'_{ij}. The probability that p'_{ij} can be inferred by Q is

$$E(p'_{ij}|\texttt{Q}) = \texttt{E}(\texttt{p}'_{\texttt{ij}}|\texttt{Q}^\texttt{p}) = 1 - \prod\nolimits_{\texttt{p}_{\texttt{ij}} \in \texttt{Q}^\texttt{p}} (1 - \texttt{s}_{\texttt{ij}}). \tag{7}$$

• **Case 2:** p'_{ij} succeeds Q (i.e., p'_{ij} succeeds one of the questions in Q. Note that p'_{ij} cannot precede a question and succeed a question because the questions in Q have no preceding/succeeding relationships). Let $\texttt{Q}^\texttt{w}$ denote the set of questions in Q that are succeeded by p'_{ij}. The probability that p'_{ij} can be inferred by Q is

$$E(p'_{ij}|\texttt{Q}) = \texttt{E}(\texttt{p}'_{\texttt{ij}}|\texttt{Q}^\texttt{w}) = 1 - \prod\nolimits_{\texttt{p}_{\texttt{ij}} \in \texttt{Q}^\texttt{w}} (\texttt{s}_{\texttt{ij}}). \tag{8}$$

• **Case 3:** p'_{ij} cannot be inferred by Q (i.e., there is no question in Q that succeeds or precedes p'_{ij}), then $E(p'_{ij}|\texttt{Q}) = \texttt{0}$. Based on $E(p'_{ij}|\texttt{Q})$, we can compute $E(\texttt{Q})$ as below.

$$E(\texttt{Q}) = \sum\nolimits_{\texttt{p}'_{\texttt{ij}}} \texttt{E}(\texttt{p}'_{\texttt{ij}}|\texttt{Q}). \tag{9}$$

With the aggregated inference expectation, we can select $\mathcal{B}$ questions. However this problem is still NP-hard and we propose two greedy algorithms in next section.

6.2 Two Greedy Algorithms

Given the inference mode, we propose two greedy question selection algorithms. The first is single question selection(SQS) algorithm, which selects one question with the maximum inference expectation. The second is multiple question selection(MQS) algorithm, which selects multiple questions with the maximum aggregated inference expectation. The first can iteratively select questions and utilize them to infer more pairs. The second selects questions in a batch and can reduce the latency.

Single Question Selection. Intuitively, in order to maximize the matching result under a given budget, we should select the pair with maximal inference expectation as the question to ask the crowd in each round. Inspired by this motivation, we propose SQS algorithm and the pseudo code is shown in Algorithm 8. The algorithm takes a partial set $(P, \prec)$ as input, and outputs a set of entity pairs as queries. It first computes each entity pair's inference expectation $E(p_{ij})$ (lines 2-3). Next, for each round of $\mathcal{B}$, we select the most beneficial question (e.g. the pair with the maximal inference expectation) into the query set Q. Then, we publish the pair as the question to the crowdsourcing platform and collect the answers from different workers. We remove and restore each pair affected by the answers and re-compute the $E(p_{ij})$ (lines 4-7). After $\mathcal{B}$ iterations, we get $\mathcal{B}$ questions. (line 8).

The time complexity of SQS is $O(\mathcal{B}n)$, where n is the number of pairs. This algorithm can adaptively produce the questions according to the response from the crowdsourcing platform effectively. However, this method may incur high the latency, because we must wait for the feedback from the crowd in each iteration. Therefore we introduce a multiple question selection algorithm.

Multiple Question Selection. Based on the definition of question selection and aggregated inference expectation, MQS aims to select a set of queries Q from P simultaneously. As the aggregated inference expectation satisfies monotonicity and submodularity as stated in Lemma 1, we can design a greedy-based algorithm with the approximation ratio of $1 - 1/e$ as proved in [29] and [8].

Lemma 1. *The aggregated inference expectation is monotone and submodular.*

Proof. Because linear combination of monotone and submodular functions is also monotone and submodular, we only need to show that the term $E(\mathsf{Q}^{\mathsf{p}})$ in case 1 is monotone and submodular.

We first prove monotonicity property. Consider two query sets $\mathsf{Q}^{\mathsf{p}}_1$ and $\mathsf{Q}^{\mathsf{p}}_2$, where $\mathsf{Q}^{\mathsf{p}}_1 \subseteq \mathsf{Q}^{\mathsf{p}}_2$. For each entity pair p_{ij} in P, we have $\prod_{p_{ij}\in \mathsf{Q}^{\mathsf{p}}_1}(1 - s_{ij}) \geq \prod_{p_{ij}\in \mathsf{Q}^{\mathsf{p}}_2}(1 - s_{ij})$. Then $1 - \prod_{p_{ij}\in \mathsf{Q}^{\mathsf{p}}_1}(1 - s_{ij}) \leq 1 - \prod_{p_{ij}\in \mathsf{Q}^{\mathsf{p}}_2}(1 - s_{ij})$. Since $|\mathsf{Q}^{\mathsf{p}}_1| \leq |\mathsf{Q}^{\mathsf{p}}_2|$, the entity pairs affected by $\mathsf{Q}^{\mathsf{p}}_2$ is no smaller than $\mathsf{Q}^{\mathsf{p}}_1$. Therefore we have $\sum_{p'_{ij}}(1 - \prod_{p_{ij}\in \mathsf{Q}^{\mathsf{p}}_1}(1 - s_{ij})) \leq \sum_{p'_{ij}}(1 - \prod_{p_{ij}\in \mathsf{Q}^{\mathsf{p}}_2}(1 - s_{ij}))$. Thus, $E(\mathsf{Q}^{\mathsf{p}}_1) \leq \mathsf{E}(\mathsf{Q}^{\mathsf{p}}_2)$ and the aggregated inference expectation is monotone.

Next, we prove the submodularity property. Consider two query sets $\mathsf{Q}^{\mathsf{p}}_1$ and $\mathsf{Q}^{\mathsf{p}}_2$, where $\mathsf{Q}^{\mathsf{p}}_1 \subseteq \mathsf{Q}^{\mathsf{p}}_2$. According to the monotonicity, $E(\mathsf{Q}^{\mathsf{p}}_1) \leq E(\mathsf{Q}^{\mathsf{p}}_2)$. If we add q_0 into the query set, it is easy to prove that $E(\mathsf{Q}^{\mathsf{p}}_1 \cup \mathsf{q}_0) \leq \mathsf{E}(\mathsf{Q}^{\mathsf{p}}_2 \cup \mathsf{q}_0)$ and $\Delta E(\mathsf{Q}^{\mathsf{p}}_1) \geq \Delta E(\mathsf{Q}^{\mathsf{p}}_2)$. Let s_0 denote the pair similarity of q_0, we have $E(\mathsf{Q}^{\mathsf{p}}_1 \cup q_0) - E(\mathsf{Q}^{\mathsf{p}}_1) = \sum_{p'_{ij}}(1 - (1 - s_0)\prod_{p_{ij}\in \mathsf{Q}^{\mathsf{p}}_1}(1 - s_{ij})) - \sum_{p'_{ij}}(1 - \prod_{p_{ij}\in \mathsf{Q}^{\mathsf{p}}_1}(1 - s_{ij})) = \Delta E(\mathsf{Q}^{\mathsf{p}}_1) + \sum_{p'_{ij}} s_0 \prod_{p_{ij}\in \mathsf{Q}^{\mathsf{p}}_1}(1 - s_{ij})$. As we know $\prod_{p_{ij}\in \mathsf{Q}^{\mathsf{p}}_1}(1 - s_{ij}) \geq \prod_{p_{ij}\in \mathsf{Q}^{\mathsf{p}}_2}(1 - s_{ij})$ and $\Delta E(\mathsf{Q}^{\mathsf{p}}_1) \geq \Delta E(\mathsf{Q}^{\mathsf{p}}_2)$, then $E(\mathsf{Q}^{\mathsf{p}}_1 \cup \mathsf{q}_0) - E(\mathsf{Q}^{\mathsf{p}}_1) \geq E(\mathsf{Q}^{\mathsf{p}}_2 \cup \mathsf{q}_0) - E(\mathsf{Q}^{\mathsf{p}}_2)$. Thus, the aggregated inference expectation is submodular. □

Based on Lemma 1, we develop a greedy-based approximation algorithm and the pseudo code is shown in Algorithm 18. MQS uses

Algorithm 3: MQS : MULTIPLE QUESTION SELECTION

```
   Input: $(P, \prec)$ : Partial order set;
   Output: Q: Query set
1  begin
2      for each entity pair $p_{ij} \in P$ do
3          compute $E(p_{ij})$;
4      Insert each node into a queue Que in descended order wrt.
         $E(p_{ij})$;
5      Q ← Que.popup;
6      $\delta = 0$;
7      for $x = 1$ to $\mathcal{B}$ do
8          for each $p_{ij}$ in Que do
9              if $E(p_{ij}) \leq \delta$ then
10                 break;
11             $\triangle E(\mathsf{Q}) = \mathsf{E}(\mathsf{Q} \cup \{\mathsf{p_{ij}}\}) - \mathsf{E}(\mathsf{Q})$;
12             if $\triangle E(\mathsf{Q}) \geq \delta$ then
13                 $\delta = \triangle E(\mathsf{Q})$;
14                 $e_{max} = p_{ij}$;
15         Q ← $e_{max}$;
16         Que.delete($e_{max}$);
17         Que.refresh();
18     return Q;
```

SQS							
Iteration	$E_{pru}(p_{11})$	$E_{pru}(p_{22})$	$E_{pru}(p_{33})$	$E_{pru}(p_{44})$	$E_{pru}(p_{25})$	$E_{pru}(p_{55})$	$E_{pru}(p_{45})$
1	~~0.54~~	1.48	~~1.2~~	~~1.29~~	1.31	2.04	1.8
2		0.96			1		

MQS							
	p_{11}	p_{22}	p_{33}	p_{44}	p_{25}	p_{55}	p_{45}
$E_{pru}(p_{ij})$	0.54	1.48	1.2	1.29	1.31	2.04	1.8
$\Delta E^{\Sigma}_{pru}(p_{ij})$		0.98			0.82		

Figure 4: Question selection process.

a queue to store $E(p_{ij})$ (lines 2-5), and δ is the max gap of inference expectation between two questions (line 6). For each iteration, the algorithm examines every pair in queue and computes the delta aggregated inference expectation. Then, the algorithm selects the entity pair with the maximum $\triangle E(\mathsf{Q})$ and inserts it into the output set. Finally, the algorithm refreshes the queue and continues to the next iteration until it reaches the given budget $\mathcal{B}$ (lines 7-18).

The most costly step in the MQS algorithm is the computation of the $\triangle E(\mathsf{Q})$ in $\mathcal{B}$ iterations. The total computational complexity is $O(\mathcal{B}|Que||E|)$. Here $|Que|$ denotes to the length of the queue and $|E|$ denotes the number of entity pairs affected by the queries. The worst-case complexity of this algorithm is also $O(\mathcal{B}n^2)$. However, this method has better performance in practice, because the $|Que|$ can be truncated by the parameter δ and the affected entity pairs are usually a small part of P.

Proposition 1. *The approximation ratio of* MQS *is* $1 - 1/e$.

Proof. Based on Lemma 1, the aggregated inference expectation satisfies monotonicity and submodularity. Then the approximation ratio of MQS algorithm is $1 - 1/e$ as shown in [29]. □

Example 3. *Given the partial order set as shown in Fig. 3, we get the pair similarity s_{ij}. According to the Equation 6, we can calculate the inference expectation of each pair in the figure. For example,*

$E(p_{55}) = 0.52 \times 3 + 0.48 \times 1 = 2.04$, and $E(p_{22}) = 0.52 \times 1 + 0.48 \times 2 = 1.48$. In the same way, we have $E(p_{11}) = 0.54$, $E(p_{33}) = 1.2$, $E(p_{44}) = 1.29$, $E(p_{25}) = 1.31$, $E(p_{45}) = 1.8$. Given $\mathcal{B} = 2$ for SQS, *in the first round, we select p_{55} as the query. We suppose the answer from the crowdsourcing platform is yes, then we prune p_{11}, p_{33}, p_{44}. p_{45} is the descendant of p_{55}. We re-compute the inference expectation for the rest pairs $E(p_{22}) = 0.96$, $E(p_{25}) = 1$. In the second round, we choose p_{25} to ask the crowds for $E(p_{25}) > E(p_{22})$. After two round of question selection, we choose p_{55} and p_{25} as the questions and find 6 alignment results. For* MQS *under the same settings, we also select p_{55} as the query at first, and p_{11}, p_{33}, p_{44}, and p_{45} can be inferred by p_{55}. Then, we compute the $\triangle E(\mathsf{Q})$ for each node. We have $\triangle E(p_{22}) = (1 - 0.48 \times 0.48) + 0.52 \times 2 + 0.48 \times 1 + (1 - 0.52 \times 0.52) - 2.04 = 0.98$, $\triangle E(p_{25}) = 2.86 - 2.04 = 0.82$. Thus, we choose p_{22}. The process is shown in Fig. 4. After question selection in one round, we choose p_{55}, p_{22} as the questions to ask the crowd together. Suppose the feedbacks from workers are all yes, then we can find 5 alignment results.*

7 ERROR TOLERANCE

As crowd workers are error-prone, we propose a quality-control strategy to overcome the bias produced by workers. We also discuss how to control the error propagation by the wrong inference.

Worker Quality Control. The most commonly used strategy is majority voting [4, 16] which is to assign each task to multiple workers, and compute the most probable result by exploiting all of its answers. Here we adopt a weighted majority voting method [12, 26] to combine the workers quality into the majority voting. First, we model the workers by the approval rate which is provided from the crowdsourcing platform or qualification test that consists of golden tasks should be accomplished by the worker before they can answer the real questions. We use the approval rate or accuracy rate to denote the worker's quality. After that, we will eliminate the low-quality workers (e.g., Worker quality < 0.7). Then, we assign the questions to qualified workers and collect the answers. A high quality worker will be assigned with a higher voting weight, and the answer with the highest weighted sum will be returned as the result. Formally, in our model, for an alignment query q and a set of workers W, a worker $w \in W$ answers the question with $\mathcal{A}_{w,q}$: $\mathcal{A}_{w,q} = 1$, if $\mathcal{A}_{w,q} = Yes$; $\mathcal{A}_{w,q} = -1$, if $\mathcal{A}_{w,q} = No$. Then, the final result of q is $\hat{r}_q = \sum_{w \in W} \omega_{w,q} \mathcal{A}_{w,q}$. If $\hat{r}_q > 0$, the answer is Yes, otherwise, the answer is No.

Error-Propagation Reduction. To reduce the error propagation, we first define the indeterministic query. A query q_i is an indeterministic query when (1) $\hat{r}_q$ is less than a specified value (e.g. $\hat{r}_q < 0.1$), or (2) error is costly (e.g. $s_{ij} < 0.4$ while $\hat{r}_{ij} > 0$), or (3) there exists one-many mapping between two *KBs* whereas the similarity of different pairs are similar (e.g. for p_{ij} and p_{ik}, $|s_{ij} - s_{ik}| < 0.1$). The rules can be added according to different *KBs*. In our framework, we propose a heuristic method, called the second review, to address the indeterministic queries. This method will produce two additional questions when encounters the answer of an indeterministic query. One is the most similar ancestor of q_i (denoted by q_a) and another is the most similar descendant of q_i (denoted by q_d). When the answer of q_i is Yes, there are four possible cases. The first one is both q_a and q_d are Yes, then we still adopt the answer of q_i to infer the partial order set. The second one, where q_a is Yes and q_d is No, is the same with the first case. The third one is both q_a and q_d are No, then we adopt the answer of q_d to infer the partial order set. The last one, where q_a is No and q_d is Yes, will be discarded or thrown out for the specified workers. When the answer of q_i is No, it goes in the similar way. By specifying appropriate thresholds, the method can effectively reduce the influence of errors. We use SQS* and MQS* to denote the improved question selection algorithms with error tolerance, and the alignment result is shown in the experiment section.

Table 1: Datasets.

Dataset	#Instance	#Classes	#Properties
YAGO	3.03M	360K	70
DBPedia	2.49M	0.32K	1.2K

Table 2: Evaluation for Partition and pruning.

RR (%)	PC (%)	F (%)
99.99	39.2	56.3

8 EXPERIMENTS

This section evaluates the performance of our approach. We first introduce our experiment settings (Section 8.1), and then conduct experiments on real-world datasets (Section 8.2). We take two state-of-the-art holistic probabilistic algorithms Paris and Pba as the machine algorithms.

8.1 Experiment Setup

Datasets. We evaluate our method on two large-scale real-world datasets *Yago* and *DBPedia*. The statistics of the datasets are shown in Table 1. The datasets and the ground truth of the alignment result can be obtained from the url: http://webdam.inria.fr/paris which were opened by the authors of Paris.

Competitors. We compare with the state-of-the-art machine algorithms Paris [33] and Pba [42] to show the improvement of alignment quality. For the human-machine comparison methods, because Power [42] can reduce the budget with 1-2 orders of magnitude without sacrificing alignment quality than other crowdsourcing methods, we choose it as the baseline. We evaluate the alignment quality using the standard metrics of precision(P), recall(R), and F1-score(F).

Crowdsourcing Platform. We use a real crowdsourcing platform ChinaCrowds[2] and publish the human-intelligence tasks (HITs) on it. Each task is a decision question, which asks workers to check whether two entities refer to the same entity. We use qualification tests to estimate worker's accuracy and eliminate low-quality workers. Each HIT contains 5 tasks and we pay each HIT 0.1\$. We combine their answers via our error tolerance strategy.

8.2 Evaluation on Real-World Datasets

Evaluating Partition and Pruning. We first evaluate the effectiveness of partition and pruning process. Here we use reduction ratio(*RR*) and pairs completeness(*PC*) as evaluating indicators. *RR* measures the ratio of entity pairs pruned by our partition and pruning process among all possible pairs. Pairs completeness is the ratio

[2]http://www.chinacrowds.com/

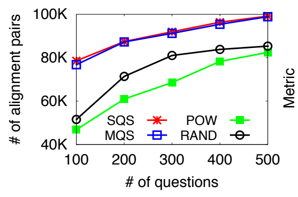
of true matches after our partition and pruning process among all true matches which is corresponding to the upper bound of recall. The result is shown in Table 2. Our partition and pruning methods show nearly perfect *RR*, whereas the *PC* is relatively low. This is because in order to reduce the matching scale, we should prune the entity pairs as many as possible. Then, the *PC* is inevitably reduced. When evaluate the question selection algorithms, we take all the matching pairs after partition and pruning process as the total matches in the following experiments.

Evaluating Question Selection Algorithms. We compare our question selection algorithms SQS and MQS with Power (POW) and random selection(RAND) algorithms. We rewrite the serial algorithm of Power based on our partition and partial order methods. To evaluate the question selection algorithms, we take all the matching pairs after partition and pruning process as the total matches in the following experiments. Figures 5(a) and 5(b)(under 500 questions) show the results. We see that with the increase of the number of questions, all the four algorithms can find more alignment pairs, because all the algorithms can infer alignment pairs on the partial order set. The alignment pairs and the quality of Power is even lower than the random algorithm because the number of unmatched pairs is much more larger than matched pairs, and the binary-search method adopted by Power will find more unmatched pairs as questions than the other methods at the beginning of the question selection. Our methods can infer more alignment pairs than the other two methods, because we can select high-quality pairs to do the inference. MQS and SQS infer nearly the same number of alignment pairs and quality because they are based on the same inference model and MQS can select as high quality pairs as SQS.

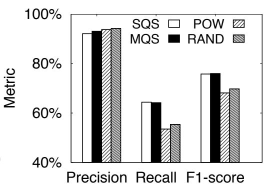
Varying Similarity Threshold θ. Parameter θ is the threshold to cluster entity pair. We evaluate the quality by varying θ and the results are shown in Figures 6(a) and (b). We can see that with the increase of θ, the precision is increased and the recall is decreased. This is because a larger θ leads to larger numbers of blocks (each block is small) and the possible alignment pairs may not be partitioned into the same block. A smaller θ leads to smaller numbers of blocks (each block is large) and many dissimilar pairs are partitioned into the same block. To make a tradeoff, we use $\theta = 0.3$ which maximizes the f1-score. $\theta = 0.3$ can also prune large amount of dissimilar entity pairs. Another observation is that SQS and MQS achieve similar quality, because MQS will not affect the quality.

Varying The Number of Questions. We evaluate our methods by varying the number of questions and the results are shown in Figures 7. We can see in Figures 7 that with the increase of the number of questions, the recall is increased whereas the precision does not change obviously. This is because if we ask more questions, we can find more similar pairs. The recall increases by about 1% when the number of questions change from 400 to 500, and the increment is less than 2‰ in the next 500 questions, because with the increase of the number of questions, the inference power is becoming lower and lower. The F1-score of SQS is a little higher than that of MQS after 200 questions, because SQS can select high-quality questions which can infer more similar pairs.

Evaluating Error-Tolerant Techniques. We evaluate our error-tolerant techniques. As shown in Figure 8, it is easy to compare two

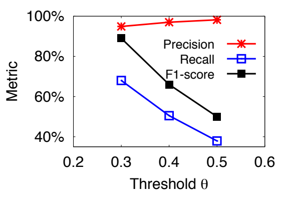


(a) Varying #questions.

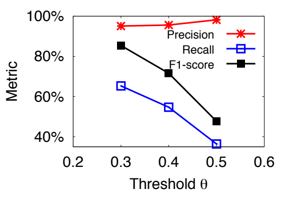


(b) Matching quality.

Figure 5: Evaluating question selection algorithms.

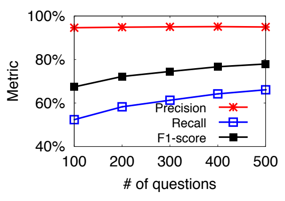


(a) SQS.

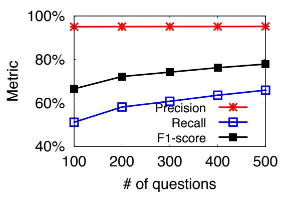


(b) MQS.

Figure 6: Alignment quality by varying the threshold θ.



(a) SQS.

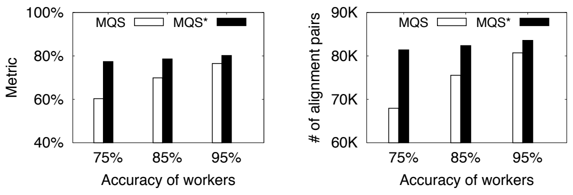


(b) MQS.

Figure 7: Alignment quality by varying #questions.

Wiki pages and thus workers from our crowdsourcing platform achieve more than 90% accuracy. To evaluate the performance of our error tolerance, we conduct a simulation experiment. We randomly generate workers with quality in 75%, 85%, and 95% based on the ground truth respectively. In order to save time and money, we only test the MQS (without error-tolerant techniques) and MQS* (with error-tolerant techniques) under 200 questions. The result is shown in Figure 8. We can see that the improved MQS algorithm with error tolerance significantly outperforms the original one under the same worker quality especially in the low worker accuracy. This is because our error-tolerant techniques can tolerate both worker errors and partial-order errors.

Comparition with state-of-the-arts. We compare with Paris, Pba, and Power. The final alignment results are shown in Table 3. It can be seen that Hike achieve as high precision as Paris and Pba because (1) Paris and Pba already achieve high precision and (2) processing unmatched pairs alignment will decrease the precision by about 1% and this part will be compensated by the process of matched pairs alignment. What is more important is that the recall is improved by about 8% under 500 questions, which means our hybrid framework will find additional 100*k* matched entity pairs with little cost. Since comprehensive considering classes, properties, and relations of entities, Paris or Pba improve about 15% recall compared to exact matching (the bottom line in the Table 3 which calculates the similarity only based on entity names), and our algorithm obtains a satisfiable quality on large-scale *KBs*. Power is 3% lower in recall than Hike under 500 questions because it will find more unmatched pairs as questions. All the elapsed time of human-machine methods in the last column exclude the latency of



(a) F1-score. (b) #alignment pairs.

Figure 8: Quality Comparison: MQS vs MQS*.

Table 3: Alignment results.

Method	P (%)	R (%)	F (%)	Time (s)
PBA	94.8	72.1	81.9	2987
PARIS	93.4	71.5	80.9	44880
POWER(500)	93.9	76.5	84.3	4384
Hike(SQS500)	94.5	**79.9**	**86.6**	3472
Hike(MQS500)	94.7	79.5	86.4	36690
Exact Matching	**95.5**	56.2	70.8	**59**

the crowd, and they are larger than machine methods because they include these machine methods in their framework. Although the SQS algorithm is more efficient, the latency of the workers makes MQS more applicable in practice.

9 CONCLUSION

In this paper, we propose a hybrid human-machine framework for entity alignment in large-scale knowledge bases. We devise a predicate-based entity partition method to partition the entities into different blocks such that we can prune the entity pairs from different blocks. To align the entities in the same block, we define a partial order set on the entities., and utilize the partial order to infer many unnecessary pairs. We formulate the question selection problem which selects $\mathcal{B}$ pairs to maximize the number of inferred pairs. We prove the question selection problem is NP-hard and propose two greedy algorithms. We develop error-tolerant techniques to tolerate errors. The experiment results on real datasets show that our framework achieves high quality and outperforms state-of-the-art approaches.

10 ACKNOWLEDGEMENT.

This work was supported by 973 Program of China (2015CB358700), NSF of China (61632016,61373024,61602488,61422205,61472198), FDCT-/007/2016/AFJ, Key Projects of Military Logistics Research (BHJ14L010).

REFERENCES

[1] N. Abdullah and R. Ibrahim. Knowledge retrieval in lexical ontology-based semantic web search engine. In *ICUIMC, Kota Kinabalu, Malaysia - January 17 - 19, 2013.*

[2] Y. Afacan and H. Demirkan. An ontology-based universal design knowledge support system. *Knowl.-Based Syst.*, 24(4):530–541, 2011.

[3] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *WWW, 2007.*

[4] C. C. Cao, J. She, Y. Tong, and L. Chen. Whom to ask? jury selection for decision making tasks on micro-blog services. *PVLDB*, 5(11):1495–1506, 2012.

[5] C. Chai, G. Li, J. Li, D. Deng, and J. Feng. Cost-effective crowdsourced entity resolution: A partial-order approach. In *SIGMOD 2016, San Francisco, CA, USA, June 26 - July 01, 2016.*

[6] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *SIGMOD 2003, San Diego, California, USA, June 9-12, 2003.*

[7] J. Fan, G. Li, B. C. Ooi, K. Tan, and J. Feng. icrowd: An adaptive crowdsourcing framework. In *SIGMOD*, pages 1015–1030, 2015.

[8] J. Fan, M. Lu, B. C. Ooi, W. Tan, and M. Zhang. A hybrid machine-crowdsourcing system for matching web tables. In *ICDE 2014, IL, USA, March 31 - April 4, 2014.*

[9] J. Feng, G. Li, H. Wang, and J. Feng. Incremental quality inference in crowdsourcing. In *DASFAA*, pages 453–467, 2014.

[10] A. Gruenheid, B. Nushi, T. Kraska, W. Gatterbauer, and D. Kossmann. Fault-tolerant entity resolution with the crowd. *CoRR*, abs/1512.00537, 2015.

[11] Q. Guo and M. Zhang. Question answering based on pervasive agent ontology and semantic web. *Knowl.-Based Syst.*, 22(6):443–448, 2009.

[12] C. Ho, S. Jabbari, and J. W. Vaughan. Adaptive task assignment for crowdsourced classification. In *ICML 2013, Atlanta, GA, USA, 16-21 June 2013.*

[13] A. Hogan, A. Polleres, J. Umbrich, and A. Zimmermann. Some entities are more equal than others: statistical methods to consolidate linked data. *NeFoRS*, 2010.

[14] W. Hu, Y. Qu, and G. Cheng. Matching large ontologies: A divide-and-conquer approach. *Data Knowl. Eng.*, 67(1):140–160, 2008.

[15] S. R. Jeffery, M. J. Franklin, and A. Y. Halevy. Pay-as-you-go user feedback for dataspace systems. In *SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 847–860, 2008.

[16] L. I. Kuncheva, C. J. Whitaker, C. A. Shipp, and R. P. W. Duin. Limits on the majority vote accuracy in classifier fusion. *Pattern Anal. Appl.*, 6(1):22–31, 2003.

[17] 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 2013, Chicago, IL, USA, August 11-14, 2013.*

[18] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer. Dbpedia - A large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[19] G. Li. Human-in-the-loop data integration. *PVLDB*, 10(12):2006–2017, 2017.

[20] G. Li, C. Chai, J. Fan, X. Weng, J. Li, Y. Zheng, Y. Li, X. Yu, X. Zhang, and H. Yuan. CDB: optimizing queries with crowd-based selections and joins. In *SIGMOD*, pages 1463–1478, 2017.

[21] G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou. EASE: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *SIGMOD*, pages 903–914, 2008.

[22] G. Li, J. Wang, Y. Zheng, and M. J. Franklin. Crowdsourced data management: A survey. *IEEE Trans. Knowl. Data Eng.*, 28(9):2296–2319, 2016.

[23] G. Li, Y. Zheng, J. Fan, J. Wang, and R. Cheng. Crowdsourced data management: Overview and challenges. In *SIGMOD*, pages 1711–1716, 2017.

[24] J. Li, J. Tang, Y. Li, and Q. Luo. Rimom: A dynamic multistrategy ontology alignment framework. *IEEE Trans. Knowl. Data Eng.*, 21(8):1218–1232, 2009.

[25] J. Li, Z. Wang, X. Zhang, and J. Tang. Large scale instance matching via multiple indexes and candidate selection. *Knowl.-Based Syst.*, 50:112–120, 2013.

[26] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Inf. Comput.*, 108(2):212–261, 1994.

[27] K. K. Lo and W. Lam. Building knowledge base for reading from encyclopedia. In *AAAI, Technical Report SS-07-06, Stanford, California, USA, March 26-28, 2007.*

[28] I. C. Mogotsi. Christopher d. manning, prabhakar raghavan, and hinrich schütze: Introduction to information retrieval - cambridge university press, cambridge, england, 2008. *Inf. Retr.*, 13(2):192–195, 2010.

[29] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions-I. *Math Program*, 14(1):265–294, 1978.

[30] L. Otero-Cerdeira, F. J. Rodríguez-Martínez, and A. Gómez-Rodríguez. Ontology matching: A literature review. *Expert Syst. Appl.*, 42(2):949–971, 2015.

[31] G. Salton, A. Wong, and C.-S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[32] C. Sarasua, E. Simperl, and N. F. Noy. Crowdmap: Crowdsourcing ontology alignment with microtasks. In *ISWC 2012, Boston, MA, USA, November 11-15, 2012, Proceedings, Part I*, pages 525–541, 2012.

[33] F. M. Suchanek, S. Abiteboul, and P. Senellart. PARIS: probabilistic alignment of relations, instances, and schema. *PVLDB*, 5(3):157–168, 2011.

[34] F. M. Suchanek, G. Kasneci, and G. Weikum. YAGO: A large ontology from wikipedia and wordnet. *J. Web Sem.*, 6(3):203–217, 2008.

[35] N. Vesdapunt, K. Bellare, and N. N. Dalvi. Crowdsourcing algorithms for entity resolution. *PVLDB*, 7(12):1071–1082, 2014.

[36] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. Crowder: Crowdsourcing entity resolution. *PVLDB*, 5(11):1483–1494, 2012.

[37] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng. Leveraging transitive relations for crowdsourced joins. In *SIGMOD 2013, New York, NY, USA, June 22-27, 2013.*

[38] S. Wang, X. Xiao, and C. Lee. Crowd-based deduplication: An adaptive approach. In *SIGMOD 2015, Melbourne, Victoria, Australia, May 31 - June 4, 2015.*

[39] S. E. Whang, P. Lofgren, and H. Garcia-Molina. Question selection for crowd entity resolution. *PVLDB*, 6(6):349–360, 2013.

[40] Y. Zheng, G. Li, Y. Li, C. Shan, and R. Cheng. Truth inference in crowdsourcing: Is the problem solved? *PVLDB*, 10(5):541–552, 2017.

[41] Y. Zheng, J. Wang, G. Li, R. Cheng, and J. Feng. QASCA: A quality-aware task assignment system for crowdsourcing applications. In *SIGMOD*, pages 1031–1046, 2015.

[42] Y. Zhuang, G. Li, Z. Zhong, and J. Feng. PBA: partition and blocking based alignment for large knowledge bases. In *DASFAA, Dallas, USA, April 16-19, 2016.*