Knowledge Base Semantic Integration Using Crowdsourcing

Rui Meng, Lei Chen, *Member, IEEE*, Yongxin Tong, and Chen Zhang

Abstract—The semantic web has enabled the creation of a growing number of knowledge bases (KBs), which are designed independently using different techniques. Integration of KBs has attracted much attention as different KBs usually contain overlapping and complementary information. Automatic techniques for KB integration have been improved but far from perfect. Therefore, in this paper, we study the problem of knowledge base semantic integration using crowd intelligence. There are both classes and instances in a KB, in our work, we propose a novel hybrid framework for KB semantic integration considering the semantic heterogeneity of KB class structures. We first perform semantic integration of the class structures via crowdsourcing, then apply the blocking-based instance matching approach according to the integrated class structure. For class structure (taxonomy) semantic integration, the crowd is leveraged to help identifying the *semantic* relationships between classes to handle the semantic heterogeneity problem. Under the conditions of both large scale KBs and limited monetary budget for crowdsourcing, we formalize the class structure (taxonomy) semantic integration problem as a *Local Tree Based Query Selection* (LTQS) problem. We show that the LTQS problem is NP-hard and propose two greedy-based algorithms, i.e., static query selection and adaptive query selection. Furthermore, the KBs are usually of large scales and have millions of instances, direct pairwise-based instance matching is inefficient. Therefore, we adopt the blocking-based strategy for instance matching, taking advantage of the class structure (taxonomy) integration result. The experiments on real large scale KBs verify the effectiveness and efficiency of the proposed approaches.

Index Terms—Knowledge base, crowdsourcing, data integration

1 Introduction

Recently, large scale knowledge bases (KBs) have been constructed and are becoming more plentiful, such as YAGO, Probase, Freebase, KnowItALL, DBpedia, NELL and DeepDive [1], [2], [3], [4], [5], [6], [7]. Since these knowledge bases are constructed independently from different sources with different techniques, different KBs usually contain overlapping and complementary information. Moreover, as knowledge acquisition is an expensive process, reusing existing KBs is strongly desirable to reduce the cost of data management. Therefore, knowledge base integration has attracted growing interests.

In the last decade, a wide variety of works have been conducted on ontology integration [8], [9], [10], [11], which is closely related to the problem of knowledge base integration, as an ontology can be treated as the conceptual system to underlie a particular knowledge base [12]. We will use the term *knowledge base* (KB) to emphasize that a KB is a physical artifact: it is a repository of information that can be accessed and manipulated in some predefined fashion. Moreover, the emerging KBs are usually of much larger scale compared to the ontology, the scale is on the order of millions of entities. Most of the existing works on ontology matching have been focused on the small ontology datasets. To integrate KBs, both data and structure information are combined to align classes, instances and relations/properties. Consider the example shown in Fig. 1, given two KBs, each KB has classes (node in circle), instances (node in rectangle) and relations/properties (edges), the alignment pairs between KBs needs to be found, each pair has an equivalence or subclass/superclass meaning. Two major subtasks in KB integration are class structure integration and instance matching. The class structure of a KB can be regarded as a *taxonomy*, for simplicity, we will use *taxonomy* in the rest of this paper. Note that relations (properties) between two knowledge bases can be manually aligned [9], as the number of relations in KBs is usually small (77 in YAGO and 1,298 in DBpedia). The first task, taxonomy integration, is a fundamental task not only in knowledge base integration but also in many on-line commercial portals which have been studied in [13], [14]. The other task, instance matching, is closely related to entity resolution, and is central to both data integration and data cleaning [9], [15], [16].

In our work, we propose a framework for KB integration considering the semantic heterogeneity of KB class structures, i.e., *semantic KB integration*. In our framework, we conduct the taxonomy integration first and then align the instances based on the taxonomy integration result. *The focus of this work is on the first step, i.e., taxonomy integration. We propose a novel crowd assisted mechanism for integrating taxonomies handling the semantic heterogeneity*. For the instance matching, we take the class information as the blocking factor and apply *data blocking* techniques [17] based on the taxonomy integration result .

- *R. Meng and L. Chen are with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Kowloon, Hong Kong, SAR China. E-mail: {rmeng, leichen}@cse.ust.hk.*
- *Y. Tong is with the State Key Laboratory of Software Development Environment, School of Computer Science and Engineering, Beihang University, Beijing 100191, China. E-mail: yxtong@buaa.edu.cn.*
- *C. Zhang is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology and Shandong University of Finance and Economics. E-mail: czhangad@cse.ust.hk.*

Manuscript received 3 Jan. 2016; revised 20 Sept. 2016; accepted 18 Oct. 2016. Date of publication 20 Jan. 2017; date of current version 30 Mar. 2017. Recommended for acceptance by G. Li.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2017.2656086

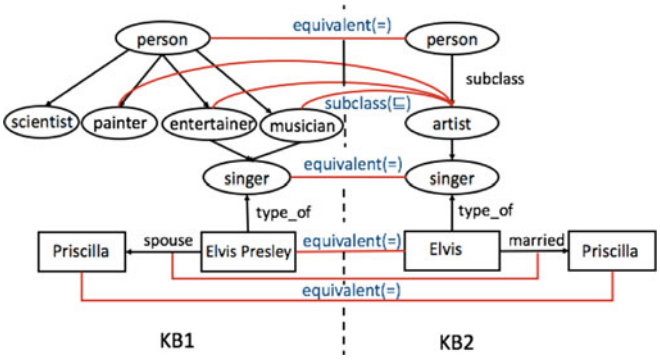


Fig. 1. Example of KB integration.

For taxonomy integration, in order to handle the taxonomy semantic heterogeneity, the *semantic* relationships between classes should be considered. In this paper, we categorize such relationships into four groups—*equivalence*, *generalization*, *specification* and *others* (none of the other three relationships holds). Existing works on ontology integration usually adopt entity resolution or classification techniques for taxonomy integration, which cannot be applied to this scope directly, since the entity resolution deals with the *equivalence* relationship and classification techniques focus on the *generalization/specification* relationship. Besides, the machine-based taxonomy integration, either suffers from the low accuracy, the state-of-the-art techniques can only achieve the 80 percent accuracy [10], [13], [18], or achieve higher accuracy but only return the super-classes or sub-classes instead of the integrated unified structure directly [8]. As shown in Fig. 2, even if we know that *artist* is a subclass of *person*, we cannot fix the position of *artist*, unless we know the relationships between the query node and all the children nodes of *person*, i.e., *entertainer*, *musician*, *painter and scientist*.

Therefore, inspired by the emerging concept of crowdsourcing for various human intrinsic tasks, such as taxonomy construction [19], entity resolution [15], language understanding [20], web wrapper generation [21], etc, we resort to the crowd to facilitate the taxonomy integration. While acknowledging the importance of human involvement in ontology integration, some works have been conducted to include user contribution, either experts or crowd workers, into the ontology integration process [22], [23]. These works limit user involvement to validating the candidate mappings, either confirm or refute a mapping pair generated by some algorithm, and do not allow users to identify the *semantic relationship* among classes, which is our focus in taxonomy integration.

Challenges. Incorporating the crowd power for taxonomy integration raises the following challenges:

1. The first challenge to utilize the crowd is to design a proper worker interface—we need to determine which type of questions should be posted to the crowd. For crowdsourced taxonomy integration, a straightforward idea is to pick one node from taxonomy T_1 and ask the *"is_Descendant"* question from the root of taxonomy T_2 in a downward manner until the node reaches the target position [24]. However, different KBs could have different granularity of class modeling, as shown in Example 1. Therefore the relative position should be context dependent, which means, we should give the crowd more information w.r.t. the context, instead of two single nodes. Consequently, how to incorporate the proper contextual information into the interface design of human intelligence tasks (HITs) should be considered.
2. Furthermore, since each crowdsourced question is associated with a monetary cost and we are usually given a limited budget. The second challenge is to construct the integrated taxonomy with the minimum number of questions. The *brute-force* approach of exhaustively posting all the possible questions to the crowd, will lead to $O(NM)$ questions in the worst case, where N and M is size of two taxonomies for integration, which is not scalable. In particular, the *brute-force* approach neglects the hierarchical structure of the taxonomy, which may help reducing the number of questions to be asked.

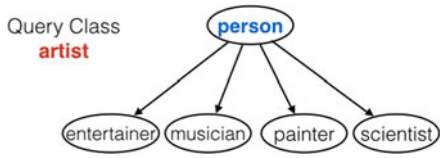


Fig. 2. Query with a subtree.

Example 1. Different KBs may have different class granularity, as shown in Fig. 1. From the classes given in Fig. 1, we can observe that class "artist" and classes "painter", "musician" "entertainer" are at different granularity, e.g., "artist" resides in a coarser-grained level w.r.t. "musician" and "painter". In this case, only considering the "equivalence" relationship between classes is not enough, we must consider all the semantic relationships: equivalence, generalization, specification and others (none of the above relationships holds). The "artist" class is more general w.r.t. the musician class and is more specific w.r.t. person class; two "singer" classes are equivalent; considering "artist" class with "scientist", the answer is others since none of the above three relationships holds between them.

To query the location of class "artist", w.r.t. the target class "person", a subtree should be provided as the contextual information to the crowd for facilitating the decision, as shown in Fig. 2.

To summarize, the contributions of our work are threefold:

- We propose a novel framework for KB integration by first conducting taxonomy integration via crowdsourcing and then performing blocking-based instance matching based on the integrated taxonomy.
- We formulate the crowd assisted taxonomy integration problem as a *Local Tree Based Query Selection* (LTQS) problem. We prove the LTQS problem is NP-hard and propose two greedy-based algorithms.
- We conduct extensive experiments on real data sets. Our experiments verify the effectiveness and efficiency of the proposed approaches.

The rest of the paper is arranged as follows. Section 2 describes some definitions and defines the KB integration problem. We introduce the models and formally define the *Local Tree Based Query Selection* problem in Section 3. Section 4 elaborates two solutions for the LTQS problem. Section 5 introduces the instance matching strategy. Section 6 illustrates the experimental studies on real KBs. We review the related work in Section 7 and finally conclude the paper in Section 8.

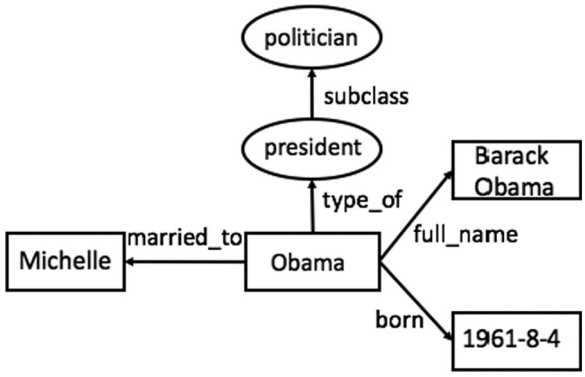


Fig. 3. An example of knowledge base.

2 Problem Statement

In this section, we review some preliminary concepts and introduce the definitions of KB integration and taxonomy integration.

A *knowledge base* is a tuple denoted by (E, L, R, P), consisting of a collection of entities E, literals L, relations R holding between entities and properties P holding between entities and literals. An entity $e \in E$ can be a class or an instance, i.e., $E = \{C \cup I\}$, where C and I represent a class set and an instance set,[1] respectively. Fig. 3 shows a toy example of KB, there are four entities-two classes, "politician" and "president" and two instances, "Obama" and "Michelle"; the date "1961-8-4" and string "Barack Obama" are literals; three relations "subclass", "type_of" and "married_to" and two properties "born" and "full_name".

In a knowledge base, the classes form a hierarchical structure, in which different classes have "subclass/superclass" relationships. If we only consider the classes in a single KB, all the classes with "subclass/superclass" (*specification/generalization*) relationships among them form a *taxonomy*. A *taxonomy*, $T = (V, \sqsubseteq)$, is the class hierarchical structure of the knowledge base, is a directed acyclic graph (DAG). The nodes $v \in V$ is a class, and $\sqsubseteq$ denotes a partial order of specification/generalization relationship between classes.

To integrate two KBs, our goal is to identify the positions of entities from one KB in another one to construct a unified KB.

Definition 1 (Knowledge Base Integration). *Given two knowledge bases, KB_1 and KB_2, knowledge base integration is the process of identifying the position of each entity from KB_1 in KB_2 (or vise versa) to get the unified knowledge base.*

KB integration has two sub-tasks: *class structure integration* and *instance matching*. Similar with KB integration, we introduce the definition of *taxonomy integration*.

Definition 2 (taxonomy integration). *Given two taxonomies, T_1 and T_2, taxonomy integration is the process of identifying the position of each class node from T_1 in T_2 to get the unified taxonomy.*

As classes refer to general concepts, there are several relationships among classes. We introduce *class semantic relationship* between classes and classify it into four categories: equivalence ($=$), generalization ($\sqsubseteq$), specification ($\sqsupseteq$) and others ($\perp$).

1. The mapping from an instance to a class can be found through "type_of" relationship.

The *equivalence* between classes refers that the two classes represent the same concept, and *specification/generalization* holds if the concept of one class is a subclass/superclass of another one, and *others* relationship refers that none of the above three relationships holds. For aligning classes, we need to consider the *class semantic relationship*. As shown in Fig. 3, we know that *politician* is a superclass of *president*, therefore, we have (*politician* $\sqsupseteq$ *president*) and (*president* $\sqsubseteq$ *politician*). For another example, given that *author* has the same meaning with *writer*, we have (*author* $=$ *writer*). For matching instances, we only need to consider the *equivalence* relationship, which holds if two instances refer to the same object in the real world, such as both *Elvis Presley* and *Cat King* refer to the American singer *Elvis*.

We adopt the *data blocking* technique which clusters the similar entities into blocks based on the classes the entities belong to. The blocks can act as filters for the entity matching, i.e., we can perform comparisons only among entities in the same block [17], or filter out any matching entity pairs from different blocks after the matching process, namely *post-filtering*. Given a class c from KB1, the similar classes from KB2 are s-hop neighbors of the correspondence of c in KB2.

Definition 3 (s-hop neighbor). *Given a taxonomy $T = (V, \sqsubseteq)$, for each node $v \in V$, its s-hop neighbor is a set of nodes, each of which can be reached from v within s-hop distance, denoted $\mathcal{N}_s(v) = \{u | dis(u, v) \leq s\}$*

Where $dis(u, v)$ is the length of shortest path consisting of "subclass/superclass" edges between nodes u and v.

To identify the positions of classes from KB1 in KB2, we need to figure out the *semantic relationships* of two classes from different KBs. For an entity from KB1, if an equivalent relationship is found in KB2, then the position is identified; otherwise, its position is determined by the position of class nodes it belongs to. Therefore, we first conduct *class structure integration* and then perform *instance matching* based on the result of unified class structure.

3 Models

As existing techniques of taxonomy integration cannot handle the problem of class position locating, we resort to the crowd to facilitate the taxonomy integration. As mentioned, due to the *semantic relationships* between classes, it is necessary to give the crowd some contextual information instead of single nodes. In this section, we introduce the crowd model adopted in our work to address the aforementioned challenges. We first introduce a *local tree based query* model in Section 3.1. Then we define the *utility* model for queries and formally introduce the *Local Tree Based Query Selection* problem in Section 3.2.

3.1 Crowd Model-Local Tree Query

As mentioned, the first challenge of utilizing crowd for taxonomy integration is how to design a proper human intelligent task (HIT) interface. Given two taxonomies T_1 and T_2, for each node in T_1 we treat it as a *query node* and the position search space consists of all nodes of T_2 (each node in T_2 is called the *target node*). For the query task, there are two issues to be addressed: on the one hand, how to model the contextual information of the target node; on the other hand, given the contextual information, how to guide the crowd to fix the position of the query node.

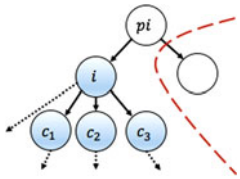


Fig. 4. Local tree structure.

Since the taxonomy may have different granularity in the class hierarchies, furthermore, as there are multiple kinds of *semantic relationships* between two classes, we need to provide the contextual information to the crowd instead of isolated nodes in the query task. We derive a novel *local tree based query*, which gives the query node and the *local tree* structure of the target node. We give multiple position choices for the crowd, besides, crowd workers can manually change the local tree structure and fix the position of the query node.

Definition 4 (Local Tree Based Query). *A local tree based query consists of a query node q and a local tree of target node t, LT_t. The local tree LT_t is a tree structure rooted at node t, $LT_t = \{t, c_1, \ldots, c_z\}$, where $C_i = \{c_1, c_2, \ldots, c_z\}$ is the children set of node t. The target of such query is to find the position of query node q w.r.t. LT_t.*

As Fig. 4 shows, the local tree consists of node i and three children $C_i = \{c_1, c_2, c_3\}$ (the right area in the red dotted curve represents the position of query node if query result is *others*). Through this structure, each query has more contextual information and hints for the crowd to make decisions. Note that the parent[2] of a target node is not included in its local tree structure as the crowd workers only need children information of target node for locating the source node's position. As the position of the query node q may be located on some edge or create a new edge in T_2, which changes the hierarchical structure of the target taxonomy. Existing approaches, such as [24], neglects the contextual information and cannot solve the aforementioned challenges, as illustrated in Example 2.

Example 2. Consider the example in Fig. 4, if we do not consider the contextual information, even if we have confirmed that q is a descendant of i and not a descendant of c_1, c_2, c_3, we cannot confirm the position of node q, instead, there are totally eight possible positions of q in this case:

- Node q is the child of i and sibling of each child node $c \in C_i$.
- Node q is the child of i and parent of one child, that is q is located on one of the edges, $e = \{(i, c_1), (i, c_2), (i, c_3)\}$, totally $\binom{3}{1}$ choices.
- Node q is the child of i and parent of two children, gives $\binom{3}{2}$ choices.
- Node q is the child of i and parent of three children c_1, c_2, c_3.

Totally, we have $(1 + \binom{nc}{1} + \binom{nc}{2} + \cdots + \binom{nc}{nc})$ possible positions, where nc is the children number of the left node, the result is eight in this example with $nc = 3$.

From the example we can see that there is a large number of possible choices for locating the query node and it is complex and tedious for the crowd to verify all the possible locations. With the *local tree based query*, we give multiple position choices for the crowd, besides, crowd workers can manually change the local tree structure and fix the position of the query node. If the node is outside LT_i, the crowd picks one answer from the *ancestor*, *descendant* and *others*; if q is inside LT_i, the crowd manually labels the position and reconstructs the local tree structure. The *ancestor* of a local tree LT_i means that q is an *ancestor* of node i; *descendant* of LT_i means that q is descendant of one or more children nodes $C_i = \{c_1, c_2, \ldots, c_z\}$, for the *descendant* choice, the crowd should explicitly state of which node q is descendant of.

3.2 Utility Function

The second challenge is how to make best use of the limited budget for crowdsourcing. We need to define some criteria to guide the query selection process. In this section, we introduce the pruning power of each query and define the *utility* function to model the usefulness of queries. The taxonomy has a hierarchical structural (*DAG* structure), which can be used to do the pruning. If node q is the descendant of t, then q is the descendant of every node in $\mathrm{ans}(t)$. Conversely, if node q is the ancestor of t, then q is the ancestor of every node in $\mathrm{des}(t)$. Note that $\mathrm{ans}(t)$ denotes the ancestor set of node t, i.e., $\mathrm{ans}(t) = \{v | v \in DAG \wedge v \sqsupseteq t\}$, and $\mathrm{des}(t)$ denotes the descendant set of node t (t included), i.e., $\mathrm{des}(t) = \{v | v \in DAG \wedge v \sqsubseteq t\}$.

Definition 5 (Candidate Pairs). *Given two taxonomies T_1 and T_2, the candidate pairs refer to the pairs that have not been pruned, which are candidates for constructing* local tree based queries, *denoted as $CP = \{(s, t) | s \in T_1 \wedge t \in T_2\}$. Each query $Q = (q, LT_i)$ must satisfy that $(q, i) \in CP$.*

Initially, all pairs $CP = \{(s, t) | s \in T_1 \wedge t \in T_2\}$ are candidate pairs, after collecting the answer of local tree based queries, we can prune the pairs according to the *pruning principle*.

Lemma 1 (Pruning Principle). *Given a pair of nodes, s from T_1 (size is N), t from T_2 (size is M) if we confirm the relationship between them, we have the following pruning rules:*

1. *If $s \sqsupseteq t$, we have $\{p \sqsupseteq q | \forall p \in ans(s) \wedge q \in des(t)\}$, the number of candidate pairs that can be pruned is: $\{|ans(s)| + 1) \cdot |des(t)|\}$.*
2. *If $s \sqsubseteq t$, we have $\{p \sqsubseteq t | \forall p \in des(s)\}$, the number of candidate pairs can be pruned is: $\{|des(s)| \cdot (M - |des(t)|)\}$.*
3. *If the answer is others, which means that s is neither ancestor nor descendant of t, denoted $s \perp t$, the number of candidate pairs can be pruned is: $\{|des(t)|\}$.*

Proof. Please refer to the Appendix A. □

Based on the *pruning principle*, we next compute the pruning power of each possible answer of the *local tree based* query. Combined with the corresponding prior probabilities, we derive the expected pruning power and take it as the *utility* of *local tree based* query.

We next describe how to model the prior belief of each situation. For the probability of a *generalization/specification* relationship, we use the overlap of instances to model the prior belief of such relationship [8]. Given two class nodes p and q, the probability that class p is a descendant of class q (equivalence included) is

2. A Taxonomy could be a tree structure or a DAG in which a class might have multiple parents.

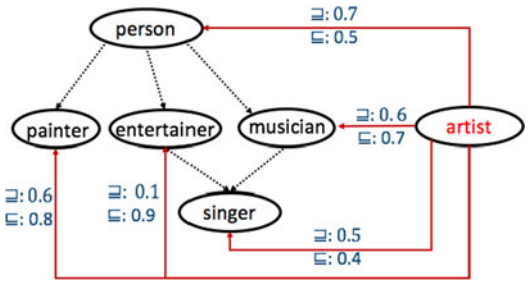


Fig. 5. KB integration with candidate alignment pairs.

$$Pr(p \sqsubseteq q) = \frac{I(p) \cap I(q)}{I(p)}. \quad (1)$$

Where $I(p)$ and $I(q)$ denote the instances of class p and q respectively. Now, we can model the prior probability of each possible result given a local tree query structure, making use of the instance information. The prior probability of q is the parent of a local tree LT_i is the probability that q is the ancestor of i, denoted

$$Pr(q \sqsupseteq LT_i) = Pr(q \sqsupseteq i). \quad (2)$$

The prior of q being the descendant of LT_i is that q is descendant of one or more child(ren) node(s) in C_i, denoted

$$Pr(q \sqsubseteq LT_i) = 1 - \prod_{c_i \in C_i} (1 - Pr(q \sqsubseteq c_i)). \quad (3)$$

As described previously, there are multiple possible positions when q is located inside the local tree LT_i. Node p could be the child of i and not parent of any child node in LT_i or p could be the child of i and parent of one or more child(ren) node(s) in C_i. The prior probability is given

$$Pr(q \in LT_i) = Pr(q \sqsubseteq i) \cdot \prod_{c_i \in C_i} (1 - Pr(q \sqsubseteq c_i)). \quad (4)$$

Besides, we have "others" choice for the position of q, which means that its outside LT_i and neither the descendant nor ancestor of LT_i, denoted $Pr(q \perp LT_i)$

$$Pr(q \perp LT_i) = (1 - Pr(q \sqsupseteq i))(1 - Pr(q \sqsubseteq i)). \quad (5)$$

We computed the prior belief of each possible result of the *local tree based query*, and after normalizing the prior belief, we can get a probability distribution of results. Note that in our dataset, the KBs use the Wikipedia identifier to represent the instances, it is easy to get the instance equivalence information (this knowledge will be hidden when matching instances); for other dataset, the equivalence can be estimated using label information of instances as adopted in [9], [25]. In work [9] the authors find initial instance matching pairs by considering the instance string representation. In work [25], the similarity of a pair of candidate entities is computed using lexical similarities between entity names.

Example 3. To illustrate the computation of prior belief of each result and the utility of each *local tree based query*, we use the example shown in Fig. 5. One KB consists of five classes, denoted KB_1, and another KB consists of a single class "artist", denoted KB_2. There are five candidate alignment pairs, each pair is associated with two scores denoting the specification probability ($\sqsubseteq$) and generalization probability ($\sqsupseteq$) w.r.t. the class "artist" to class from KB_2.

TABLE 1
Prior Distribution

Local Tree Based Query	$\sqsupseteq$	$\sqsubseteq$	$\in$	$\perp$
$(artist, LT_{person})$	0.31	0.53	0.06	0.1
$(artist, LT_{painter})$	0.54	0	0.4	0.06
$(artist, LT_{entertainer})$	0.59	0.32	0.03	0.06
$(artist, LT_{musician})$	0.43	0.31	0.2	0.07
$(artist, LT_{singer})$	0.33	0	0.41	0.25

Given the specification/generalization probability, we can compute the prior belief of each local tree based query. For (artist, LT_{person}), the prior of "artist" being the ancestor of the local tree is: $Pr(artist \sqsupseteq LT_{person}) = 0.5$; the prior of being the descendant is: $Pr(artist \sqsubseteq LT_{person}) = 1 - (1 - 0.6) * (1 - 0.1) * (1 - 0.6) = 0.856$; the prior of located in the local tree is : $Pr(artist \in LT_{person}) = 0.7 * (1 - 0.6) * (1 - 0.1) * (1 - 0.6) = 0.1$; last, the prior of neither descendant nor ancestor is : $Pr(artist \perp LT_{persion}) = (1 - 0.7) * (1 - 0.5) = 0.15$. Then, we normalize the scores of each situation and get the prior distribution. The prior distribution of each candidate local tree based query is summarized in Table 1.

Based on the prior distribution, we can compute the expected pruning power of each local tree based query, denoted $Pru(Q)$. For query $(artist, LT_{person})$, the numbers of alignment pairs being pruned are 5,1,5,5, respectively, when the answer being $(\sqsupseteq, \sqsubseteq, \in, \perp)$. The utility is: $Pru((artist, LT_{person}))$= 0.31 * 5 + 0.53 * 1 + 0.06 * 5 + 0.1 * 5 = 2.88.

Aggregate Utility. We next discuss the *aggregate utility* of a query set, which is the expected pruning power of a query set. Let $\mathcal{Q}$ denote the query set, $|\mathcal{Q}| = k$, and $\mathcal{A}$ denote all the possible answer set for $\mathcal{Q}$, the *Cartesian* product of answer set of each query, denoted: $\mathcal{A} = A_1 \times A_2 \times \cdots \times A_k$. Since each A_i consists of 4 answers, $\mathcal{A}$ has 4^k results, denoted as $\mathcal{A} = \{R_1, R_2, \ldots, R_{4^k}\}$. Each element R_i in the result set consists of k answers, $R_i = \{a_1, a_2, \ldots, a_k\}$. For each $Q_i \in \mathcal{Q}$, each possible result a_i of Q_i would have some pruning power which would prune some candidate pairs. We say that cp_{ij} has been pruned by a_i if node j is pruned in the candidate set of node i. And a pair can be pruned if at least one query have the prune power on it. Given the result set $R_p \in \mathcal{A}$, the pruning power on candidate pair cp_{ij} is

$$Pru(cp_{ij}|R_p) = 1 - \prod_{a_s \in R_p} (1 - Pru(cp_{ij}|a_s)), \quad (6)$$

where, the $Pru(cp_{ij}|a_s)$ is the pruning power of a_s on pair cp_{ij}

$$Pru(cp_{ij}|a_s) = \begin{cases} 1 & if\ cp_{ij} \in Ar(a_s) \\ 0 & otherwise, \end{cases} \quad (7)$$

where $Ar(a_s)$ is the area that will be affected, a pair cp is in the affected area if it can be pruned by answer a_s. The detail of computing pruning power of each answer result of each *local tree based* query is illustrated in Section 4.3.

TABLE 2
Summary of Notations

Notation	Description
LT_i	Local tree rooted at entity i
$\mathcal{C}_i$	children set of node (entity) i
$des(e)$	Descendants of entity e
$ans(e)$	Ancestors of entity e
cp_{ij}	Candidate pair with entity i from KB1 and j from KB2
Q	Local tree based query
$\mathcal{A}$	Possible answer sets for a given query set
R	An element in answer sets
a	The answer of a given query
$Ar(a)$	The nodes that will be affected by the answer a
$Cand(e)$	The candidate entities of entity e
$Pru(Q)$	The pruning power of query Q
$Pr(a)$	The prior belief of result a occurs
$U(\mathcal{Q})$	The utility function of a query set $\mathcal{Q}$

Next, we define the *utility* of a query set $\mathcal{Q}$

$$\begin{aligned} U(\mathcal{Q}) &= \sum_{R_p \in \mathcal{A}} Pr(R_p) \cdot \sum_{cp_{ij} \in CP} Pru(cp_{ij}|R_p) \\ &= \sum_{cp_{ij} \in CP} \sum_{R_p \in \mathcal{A}} Pr(R_p) \cdot Pru(cp_{ij}|R_p). \end{aligned} \tag{8}$$

Note that the computation for the aggregate utility of a query set $\mathcal{Q}$ given in Equation (8) needs to traverse all the possible answer results in $\mathcal{A}$ which size is $4^{|\mathcal{Q}|}$. Therefore, the computation becomes intractable if the size of $\mathcal{Q}$ grows large. Next, we derive another formula to compute the aggregate utility defined in Equation (8)

$$U(\mathcal{Q}) = \sum_{cp_{ij} \in CP} 1 - \prod_{Q_i \in \mathcal{Q}} 1 - Pru(cp_{ij}|Q_i), \tag{9}$$

where $Pru(cp_{ij}|Q_i)$ is the expected probability of candidate pair cp_{ij} being pruned by Q_i, denoted

$$Pru(cp_{ij}|Q_i) = \sum_{i}^{4} Pr(a_i) \cdot Pr(cp_{ij}|a_i). \tag{10}$$

Lemma 2. *The aggregate utility functions defined by Equations (8) and (9) are equivalent.*

Proof. Please refer to the Appendix B. □

By the Equation (9), we can accelerate the utility computation by keeping the production value $\prod_{Q_i \in \mathcal{Q}}(1 - Pru(cp_{ij}|Q_i))$ for each query set $\mathcal{Q}$. In this case, to compute the utility value of a new query set by adding a query Q, we only need to traverse the CP set and compute the $Pru(cp_ij|Q)$. After adding a new query, we will update the production value. Therefore the computation of aggregate utility of a given set can be done in $O(|CP|)$ time.

Based on the definition of *aggregate utility* of a set of *local tree based queries*, we formally introduce the *Local Tree Based Query Selection problem*.

Definition 6 (Local Tree Based Query Selection). *Given two taxonomies, T_1 and T_2, a set of candidate pairs $CP = \{(s,t)|s \in T_1, t \in T_2 \wedge t \in Cand(s)\}$. Each local tree based query Q consists of a query node q and a local tree LT_i, where $(q,i) \in CP$. Local tree based query selection problem is to select a set of local tree based queries, $\mathcal{Q}$, which gives the maximum utility value U under a limited crowdsourcing budget k (total number of Human Intelligence Tasks, a.a HITs).*[3]

Note that, in our model all query nodes are from source taxonomy T_1, target nodes and local tree structures are from target taxonomy T_2. Given two taxonomies, we select the taxonomy with larger size and finer granularity as the target taxonomy and locate positions of the nodes from source taxonomy in the target taxonomy.

For brevity, a summary of notations used in this paper is shown in Table 2

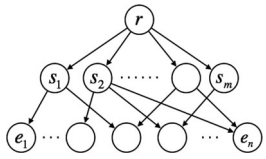


Fig. 6. NP-hard proof for query selection problem.

4 Query Selection Algorithms

In this section, we first prove that *LTQS* problem is NP-hard in Section 4.1; then we propose two greedy-based algorithms for *LTQS* problem.

4.1 Hardness Analysis

Theorem 1. *Given two taxonomies T_1 and T_2 and a integer k, select k local tree based queries that maximize the utility is NP-hard.*

Proof. We prove that the *LTQS* problem is NP-hard by a reduction from the *Maximum Coverage Problem*.

An instance of maximum coverage problem is: A set $\mathcal{U}$ of n elements, a collection of $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ of m subsets of $\mathcal{U}$, such that $\bigcup_i S_i = \mathcal{U}$, the goal is to select k subsets from $\mathcal{S}$ such that their union has the maximum cardinality. We construct an instance of two taxonomies: T_1 has a single node q, and T_2 is a DAG with three levels: a root node r at level 0, level 1 is consists of $m(m \geq k)$ children nodes of r, and level 2 consists of n nodes, which are children of nodes in level 1, illustrated in Fig. 6. The prior of q is given as follows: $Pr(p \sqsubseteq r) = 1$, for each $s \in level(1)$, $Pr(p \sqsupseteq s) = 1$ and for each $e \in level(2)$, $Pr(p \sqsupseteq e) = 1$. This means that the query node q should be located between level 0 and level 1.

In this case, according to the pruning principle, the pruning power of selecting the query rooted at r or rooted at any node in level 2 is 1, and the pruning power of selecting the query rooted at each node in level 1 is equal to the number of its children plus itself. (≥ 1). In order to maximize the pruning power, the node in level 1 is a better choice than any nodes in level 0 or level 2, so the best strategy is to select the k nodes all from level 1. In this instance, we have each node in level 1 corresponds to a set S in the *maximum coverage problem*, and each node in level 2 corresponds to an element e. The *aggregate* utility of a query set is the number of nodes that has been pruned.

Therefore, *local tree based query selection* picks query set corresponding to the set such that the maximum of nodes, which corresponds to the elements, are covered.

3. In our work, all HITs have the same price, therefore, k is proportional to the monetary budget.

Conversely, each solution of the *Local Tree Based Query Selection Problem* is a solution of the *Maximum Coverage Problem*. □

Since the LTQS problem is NP-hard, we derive two greedy algorithms: A static query selection problem and an adaptive query selection algorithm, which are discussed in the following sections.

4.2 Static Query Selection

Theorem 2. *The utility function of a query set $\mathcal{Q}$ defined in Equation (8) is monotone.*

Proof. Consider two query sets $\mathcal{Q}_1$ and $\mathcal{Q}_2$, where $\mathcal{Q}_1 \subseteq \mathcal{Q}_2$. For each result $R_p^1 \in \mathcal{A}_1$, we can find a collection of supersets of R_p^1, denoted as $Sup(R_p^1) = \{R_q^2 | R_q^2 \supseteq R_p^1 \wedge R_q^2 \in A_2\}$. Also, we have $P(R_p^1) = \sum_{R_q^2 \in Sup(R_p^1)} P(R_q^2)$. From Equation (6), it is obvious that $Pru(cp_{ij}|R_p^1) \leq Pru(cp_{ij}|R_q^2)$ if $R_p^1 \subset R_q^2$. Therefore, we have the utility of $\mathcal{Q}_2$ as follows:

$$\begin{aligned} U(\mathcal{Q}_2) &= \sum_{cp_{ij} \in CP} \sum_{R_p^1 \in \mathcal{A}_1} \sum_{R_q^2 \in Sup(R_p^1)} P(R_q^2) \cdot Pru(cp_{ij}|R_q^2) \\ &\geq \sum_{cp_{ij} \in CP} \sum_{R_p^1 \in \mathcal{A}_1} \sum_{R_q^2 \in Sup(R_p^1)} P(R_q^2) \cdot Pru(cp_{ij}|R_p^1) \\ &= \sum_{cp_{ij} \in CP} \sum_{R_p^1 \in \mathcal{A}_1} Pru(cp_{ij}|R_p^1) \cdot \sum_{R_q^2 \in Sup(R_p^1)} P(R_q^2) \\ &= \sum_{cp_{ij} \in CP} \sum_{R_p^1 \in \mathcal{A}_1} Pru(cp_{ij}|R_p^1) \cdot P(R_p^1) \\ &= U(\mathcal{Q}_1). \end{aligned} \tag{11}$$

Based on Equation (11), we state that utility of query set is monotone. □

Theorem 3. *The utility function of query set has the property of submodularity. That is, given two query sets $\mathcal{Q}_1, \mathcal{Q}_2$ and a query Q_t, where $\mathcal{Q}_1 \subset \mathcal{Q}_2$, we have $U(\mathcal{Q}_1 \cup Q_t) - U(\mathcal{Q}_1) \geq U(\mathcal{Q}_2 \cup Q_t) - U(\mathcal{Q}_2)$.*

Proof. Please refer to the Appendix C for the proof. □

Algorithm 1. Static Query Selection Algorithm

Input: Two taxonomies T_1, T_2, query budget k,
Output: Query set $\mathcal{Q}$

```
1  Initialize Q ← ∅ and candidate pairs CP according to Definition 5
2  Candidate query set CQ ← ∅
3  for each candidate pair cp_ij ∈ CP do
4      generate local tree based query Q = {i, LT_j}
5      CQ ← CQ ∪ Q
6  for s = 1 to k do
7      for each candidate query Q ∈ CQ do
8          ΔU(Q) = U(Q ∪ Q) − U(Q)
9      Q* = arg max_{Q∈CQ} ΔU(Q)
10     Q ← Q ∪ Q*
11     CQ ← CQ − Q*
12 return Q
```

Based on Theorems 2 and 3, we propose a greedy-based approximation algorithm, *static query selection* algorithm, to select the k local tree based queries, as shown in Algorithm 1. In each iteration, the algorithm selects the query Q which gives the maximum delta utility (the increase in utility after adding this query to the selection set) and adds it to the selection set $\mathcal{Q}$, until the set size reaches the given budget k.

Lemma 3. *The static query selection algorithm shown in Algorithm 1 achieves an approximation ratio of $1 - \frac{1}{e}$.*

Proof. Based on the monotone and submodular properties of the aggregate utility of a *local query based query* set, the approximation ratio of the greedy algorithm illustrated in Algorithm 1 is $1 - \frac{1}{e}$ [26]. □

Computational Complexity Analysis. Initialization operation of Algorithm 1 takes $O(|CP|)$ time (lines 1∼5); the algorithm has k iterations (lines 6∼11); in each iteration, the algorithm first computes delta utility of each query (lines 7∼8), the computational complexity is $O(|CQ| \cdot |CP|)$ (using the formula in Equation (9)); the max operation (line 9) is $O(|CQ|)$. Therefore, the complexity of Algorithm 1 is $O(k \cdot |CQ| \cdot |CP|)$.

4.3 Adaptive Query Selection

As the answer of each query is uncertain, we could only get the answer after the query task is finished by the crowd workers. The pruning strategy is applied according to the outcome of past queries, which may further influence the future query selection. Therefore, a better query selection strategy is to adaptively make a sequence of decisions.

In this section, we propose a greedy-based *adaptive* algorithm for *Local Tree Based Query Selection* problem. Suppose we select k queries, $Q_1, Q_2, \ldots, Q_k$ in a way that the choice of Q_i depends on the query results of $Q_1, Q_2, \ldots, Q_{i-1}$. Initially, we have all candidate queries CQ, in each iteration, $Q_i = \{a, LT_b\}$ has four possibilities, each of them has some pruning power and is associated with a prior probability. The prior probability modeling is given in Section 3.2.

Next, we discuss the pruning power of each situation. The $q \sqsupseteq LT_i$ and $q \perp LT_i$ can be derived by the *pruning principle*, which is equivalent with $q \sqsupseteq i$ and $q \perp i$. Then, we discuss the pruning power of $q \sqsubseteq LT_i$ and $q \in LT_i$. If q is inside the LT_i, then the crowd can label the position of q, besides, $\mathrm{ans}(q)$ must not be descendant of each $c_i \in C_i$ and $\mathrm{des}(q)$ also should be the descendant of i, therefore, the number of nodes that can be pruned is

$$\begin{aligned} &Pru(q \in LT_i) \\ &\quad = (|Cand(q)| - 1) + \sum_{a \in \mathrm{ans}(q)} |Cand(a) \backslash \mathrm{des}(LT_i)| \\ &\qquad + \sum_{b \in des(q) \backslash q} |Cand(b) \backslash \mathrm{ans}(i)|, \end{aligned} \tag{12}$$

where, $\mathrm{des}(LT_i) = \bigcup_{c_i \in C_i} \mathrm{des}(c_i)$. Note that, given a local tree LT_i, we can get $C = \{v | v \in child(i)\}$. If q is the descendant of LT_i, then it could be the descendant of one or more nodes in C_i. Our query requires the crowd to label which nodes are ancestors of query node q, therefore the number of pairs being pruned is

$$Pru(q \sqsubseteq LT_i) = \sum_{c_i \in C_i} \frac{Pr(q \sqsubseteq c_i)}{Pr(q \sqsubseteq LT_i)} \sum_{a \in \mathrm{des}(q)} |Cand(a) \backslash \mathrm{des}(c_i)|. \tag{13}$$

Based on the above discussion, we have covered all the possible results of *local tree based query* and modeled the prior

TABLE 3
Static Query Selection

Local Tree Based Query	$U(Q\|\emptyset)$	$U(Q\|\mathcal{Q})$
$(artist, LT_{person})$	2.88	0.87
$(artist, LT_{painter})$	2.27	0.9
$(artist, LT_{entertainer})$	2.77	1.12
$(artist, LT_{musician})$	3.06	-
$(artist, LT_{singer})$	2.82	1.08

and pruning power. The *utility* of a query $Q_i = \{a, LT_b\}$ is defined as the expected pruning power, denoted

$$U(Q_i|Q_1, Q_2, \ldots, Q_{i-1}) = \sum_{l \in A_b} \frac{Pr(l)}{D} \cdot Pru(l). \quad (14)$$

where, A_b is the result set, $A_b = \{a \sqsupseteq LT_b, a \sqsubseteq LT_b, a \perp LT_b, a \in LT_b\}$, and D is the normalization factor, $D = \sum_{l \in A_b} Pr(l)$.

The *utility* defined in Equation (14) computes the expected increased pruning power of a single *local tree based query* Q_i conditioned on the observed results of queries that have already been selected. The adaptive selection strategy is that each time we select the query Q_i, which has the largest increase in *utility* value, $U(Q_i|Q_1, Q_2, \ldots, Q_{i-1})$; then collect answer from the crowd and do the pruning step according to the answer. If all positions in the source taxonomy T_1 are fixed or the selection reaches the budget limit, then stop, otherwise repeat the selection procedure. The *adaptive* selection algorithm is shown in Algorithm 2.

Computational Complexity Analysis. Similar with the Algorithm 1, the computational complexity of Algorithm 2 is $O(k \cdot |CQ| \cdot |CP|)$.

Example 4. Next, we illustrate the two proposed algorithms with the same input as shown in Example 3. Suppose the input budget is 2 ($k = 2$). For static query selection, we first compute the utility of each local tree based query. As shown in Example 2, the utility can be computed using the prior distribution and pruning power of each answer result. The static query selection algorithm process is given in Table 3.

As shown in the table, in the first iteration, we select $(artist, LT_{musician})$ into the query set $\mathcal{Q}_{static}$. Then, in the second iteration, we compute the delta utility of each query w.r.t. the query set $\mathcal{Q}_{static} = \{(artist, LT_{musician})\}$. Then we select $(artist, LT_{entertainer})$ into the set. The final result is $\mathcal{Q}_{static} = \{(artist, LT_{musician}), (artist, LT_{entertainer})\}$.

For the adaptive query selection, we will also select $(artist, LT_{musician})$ in the first iteration, then we will collect the answer of this query from the crowd. The collected answer is "ancestor" and based on this answer we will prune two candidate pairs: $(artist, musician)$ and $(artist, singer)$. In the second iteration, we recompute all the utilities, and select the query with maximum utility value which is $(artist, LT_{person})$ of utility value 1.87. The final result is $\mathcal{Q}_{adaptive} = \{(artist, LT_{musician}), (artist, LT_{person})\}$.

Compare the two query selection results, the position of "artist" can be fixed by $\mathcal{Q}_{adaptive}$ since when querying the $(artist, LT_{person})$ query, it is located in the local tree. However, with the query set $\mathcal{Q}_{static}$ we can not fix the position as both answers are "ancestor" for the two queries.

Algorithm 2. Adaptive Query Selection Algorithm

Input: Two taxonomies T_1, T_2, query budget k,
Output: Query set $\mathcal{Q}$
Initialize $\mathcal{Q} \leftarrow \emptyset$ and candidate pairs CP according to Definition 5
Candidate query set $CQ \leftarrow \emptyset$
for *each candidate pair* $cp_{ij} \in CP$ **do**
    generate local tree based query $Q = \{i, LT_j\}$
    $CQ \leftarrow CQ \cup Q$
for $s = 1$ to k **do**
    for *each candidate query* $Q \in CQ$ **do**
        Compute $\Delta U(Q|\mathcal{Q})$ according to Equation (14)
    $Q^* = \arg\max_{Q \in CQ} \Delta U(Q|\mathcal{Q})$
    $\mathcal{Q} \leftarrow \mathcal{Q} \cup Q^*$
    $CQ \leftarrow CQ - Q^*$
    Observe result of query Q
    $a \leftarrow answer(Q)$
    Update CP and CQ influenced by the answer $Ar(a)$
return $\mathcal{Q}$

5 Blocking Based Instance Matching

In this section, we consider aligning a large amount of instances in knowledge bases. Since each instance of a KB refers to an object in the real world, instance alignment is to find the correspondence relationship between two instances from different KBs and interlink them. For matching instances between KBs, we regard it as a *Clean-Clean ER* problem, which is the process of detecting pairs of matching entities among two large, clean but overlapping collections of entities. This follows the assumption that there is no duplication in each KB (if duplications exist, de-duplication techniques [8], [9] can be applied first).

Large KBs have millions of instances, e.g., Freebase contains 13 million entities, and Probase has 16 million entities. Naive pairwise based instance matching is intractable for matching entities between two large KBs. In order to scale to large volumes of data, approximate techniques are adopted, among which *data blocking* is the most prominent. *Data blocking* techniques cluster the similar entities into blocks and perform comparisons only among entities in the same block [17]. In our instance matching problem between KBs, we consider the class as the blocking factor. As mentioned in Section 1, we do not need to compare an entity in "animal" to those entities in "people". Therefore, we can take several similar classes as a cluster. In our work, given a class c from KB1, the similar classes from KB2 are s-hop neighbors of the correspondence of c in KB2.

For the *blocking based* instance matching, how to define the size of the cluster is a core problem, and there is a trade off between accuracy and recall. A smaller cluster size has a stronger blocking power on instance pairs, but it would impair the recall since some matched pairs may be neglected; in another hand, adopting a larger range would introduce some false positive pairs and increase the computing complexity. In this section, we use s-hop neighbor for class clustering, where the s is a parameter set through experimental study. In our work, we adopt a fixed s over all classes, the reasons are twofold: First, there could be thousands of classes in a KB, it is not trivial to find the "best" blocking size for each class; second, according to our experimental study, when s reaches 3, the filtering power

TABLE 4
Statistics of YAGO, DBpedia

Dataset	#Facts	#Instances	#Classes	# Relations
YAGO	26,120,106	2,747,873	292,898	77
DBpedia	26,770,236	2,526,321	327	1,298

has decreased significantly and does not have much difference with instance matching result without filtering.

For each class c_i, the block consists of all its s-hop neighbors, denoted $Block(c_i) = \{c_j | c_j \in \mathcal{N}_s(c_i)\}$. For an instance $e \in c_i$, we only compare it with those entities belong to the classes in the block of c_i. All other comparisons are neglected, then we adopt existing techniques for *entity resolution*, e.g., string similarity and similarity flooding [27], to compute the similarities between entities and for each entity select the candidate with maximum similarity score as the matched one.

6 Experimental Study

6.1 Setup

Real Knowledge Bases. We choose to align two large scale KBs, YAGO [1] and DBpedia [5]. The KBs are represented as text files containing a list of triples of facts. Our experiments perform taxonomy integration and instance matching. The statistics of YAGO and DBpedia are given in Table 4. Both KBs have several million instances and thousands of classes. For the instance of the KBs, both of them use Wikipedia identifiers to describe instances, therefore, the ground truth for instance matching can be easily obtained (this is ignored during the instance matching process); for the taxonomy integration task, we check the accuracy of class alignment using the gold standard offered by PARIS [8]. Note that this gold standard is incomplete and can serve only for precision, and compare the performance of our algorithms with existing approaches [8].

6.2 Crowdsourcing Platform

In our experiments, we use gMission [28] as the platform for conducting crowdsourcing tasks. The *local tree based query* is designed as a single choice question and workers are asked to identify the relationships between the query node and target node w.r.t. the *local tree* structure of the target node, the human intelligence task interface is shown in Fig. 7.

As observed from Fig. 7, each HIT has five choices in terms of the semantic relationship between query node and target node, i.e., *ancestor*, *equivalent*, *inside the local tree*, *descendant of the local tree* and *others*. Specifically, if the answer is *inside the local tree*, workers are requested to pick the node(s) that is (are) descendant of the query node; or if the answer is *descendant of the local tree*), workers need to pick the node that is the ancestor of the query node. For example, we found that *Musical Festival* is located in the *local tree* and is the parent of *Blues_Festivals* and *Classical_music_festivals*. Following the usual practice of ensuring the quality when we use crowd [27], we design a qualification test consists of the golden pairs in our dataset, only users passed the task are allowed to do the tasks. We assign each HIT to seven workers and aggregate the answers by majority voting strategy.

6.3 Taxonomy Integration

Preprocessing. Instead of assigning each class in the taxonomy of one Knowledge Base to multiple classes in the taxonomy

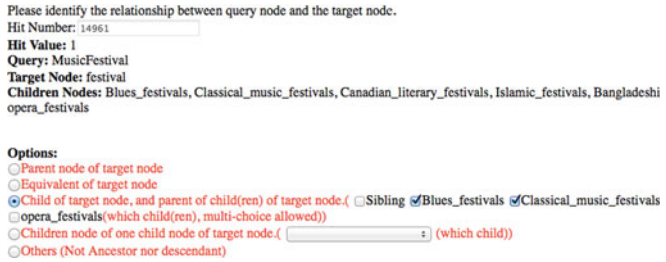


Fig. 7. Human intelligence task (HIT) interface.

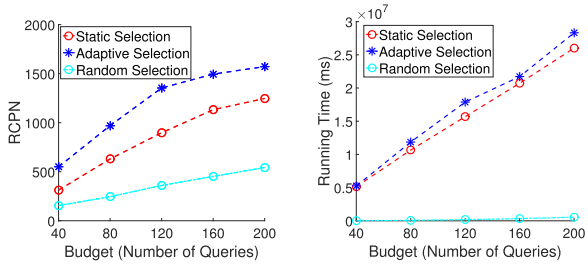
of another KB, The target of our taxonomy integration is try to fix the position of each class. Given the input two KBs, we take one KB as the source and another one as the target. In the brute-force manner, for each class of source KB, all the classes in the target KB are considered. However, such approach has the complexity of $O(MN)$ (M, N are class numbers of KBs), which is a waste of effort since most of the classes are not relevant. Also, it is intractable and costly for the crowd to conduct all those tasks, i.e., totally $327 * 292{,}898 = 95{,}777{,}646$ tasks in our dataset. Therefore, for each class c from a taxonomy, we first reduce the candidate list size of c by computing a prior belief of *generalization/specification* relationships and filter the classes with prior score lower than a threshold θ. Any method for computing the prior score can be applied here, in our work, similar with [8], we use the instance overlap to compute the prior score (as the instances are both represented using Wikipedia identifiers, we use this information for computing).

Parameters. In our dataset, we treat the DBpedia as the source KB and YAGO as the target KB. With the filter threshold $\theta = 0.01$, for each class c_i from source KB, class node c_j in target KB that with an *generalization/specification* ($c_i \sqsubseteq c_j$) relationship belief score below the threshold are filtered out. According to [8], the influence of θ on final *generalization/specification* score is trivial and could be neglected, therefore, for a higher coverage, we fix the θ value to 0.01 here. Totally 2,089 pairs are remained (|CP| = 2,089) after the filtering step (the pairs with $Pr(c_1 \sqsubseteq c_2) < \theta$ are filtered out). We vary the number of queries from 2∼10 percent (40∼200) of the total candidate pairs (|CP|).

Quality Control. As crowd workers may have errors, we assign each task to multiple workers to ensure the answer quality. For each task assigned to m workers with a_1, $a_2, \ldots, a_p$ choices. For choice a_i, if t workers vote for it, the confidence of this answer is $\frac{t}{m}$. For each task, we use majority voting to get the choice answer. To ensure answer quality and prevent error propagation, we only accept the answer and perform pruning when the majority answer has a high confidence, i.e., larger or equal than $\frac{\lceil \frac{m}{2} \rceil}{m}$ (more than half of the workers vote for this choice). In our experiment, we find that with the contextual information given in the *local tree based query* task, with various budget, on average 92 percent of the task answers have high confidence.

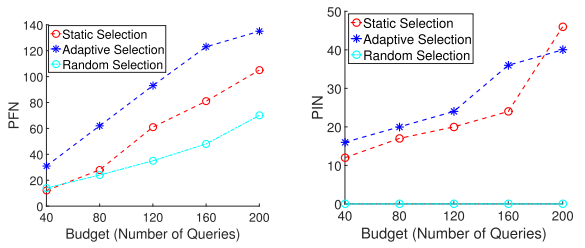
Answer Aggregation. In our experiment, we assign each HIT to seven workers, each worker will give an answer among the given five choices, shown in Fig. 7. First, we use majority voting to get the choice number. If the choice is *ancestor of the local tree*, *equivalent with the target node* or *others*, we just return the choice by majority voting; if the answer is *inside the local tree* or *descendant of the local tree*, the answer includes the corresponding children nodes that are the

Fig. 8. Utility comparison.



(a) Reduced Candidate Pair (b) Running Time

Fig. 9. Query selection algorithm evaluation.



(a) Class Position Fixed (b) Class Position Improved

Fig. 10. Taxonomy integration comparison.

descendant or is the parent of the query node. We unify the answers from different workers as our final answer.

Evaluation Metrics. For the query selection approach evaluation, we implemented three query selection algorithms: Static query selection, adaptive query selection, and random query selection. We take the random query selection algorithm, which randomly selects the query in each selection, as the baseline and compare the performances of these three algorithms using the following metrics: The utility of selected query set, reduced candidate pair number (RCPN), the precision of alignment and the running time. For the taxonomy integration evaluation, to compare our integration result with other techniques (we compare our approach with state-of-the-art approach, PARIS [8]), we use the following metrics: Number of classes that the position have been fixed (PFN), number of classes that the position have been improved (PIN). Here, we consider that the position of a class is fixed if the class is located inside the given local tree or equivalent to the target node; the position of a class is improved if it is assigned to a more specific class (if the choice is *descendant of the local tree*) or a certain *generalization/specification* ($\sqsubseteq$) relationship is corrected by the crowd (if the choice is *ancestor of the local tree*).

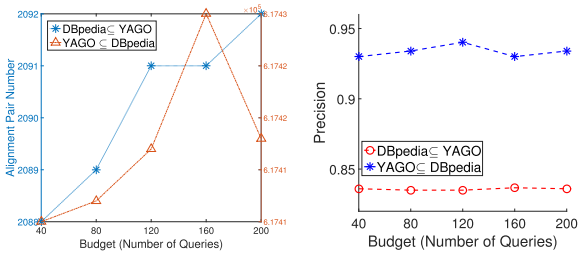
The utilities value of different selection strategies are shown in Fig. 8, we only compare the static selection algorithm with the random selection algorithm. The reason that we do not compare the utility value of the adaptive selection algorithm is that it will do the pruning each step which will change the utility value. From the result we can observe that static selection algorithm outperforms the random selection strategy.

The results of reduced candidate pairs number and running time of three query selection algorithms are illustrated in Fig. 9. We can observe that the adaptive query selection algorithm outperforms the static and random strategy, in terms of RCPN. As the adaptive selection algorithm will do the selection based on previous results, which avoids unnecessary queries. For the running time, adaptive selection strategy takes a slight more time than static selection. This difference is caused by the time spent in the adjusting and pruning step, as the adaptive query selection algorithm will adjust the input and do the pruning according to the answer at each iteration. Furthermore, to compare with other existing taxonomy integration approach, i.e., PARIS, which assigns multiple classes of the target KB for each class from source KB, we record number of classes that the positions have been fixed and number of classes that the positions have been improved through our algorithms, shown in Fig. 10. From the comparison results, we can see that the adaptive query selection algorithm achieves the best performance among the three algorithms, for both PFN and PIN metric.

Statistics of Integration Result. For the class integration, we get the result according to the query results and aggregate the answers from different workers. The results consist of a set of *generalization/specification* pairs (class alignment pairs). We evaluated the precision of the class alignment, i.e., the generalization relationship pairs ($\subseteq$). Some *generalization/specification* pairs, together with the notion TRUE or FALSE are provided by PARIS.[4] We use this for our precision evaluation. The alignment pair number (APN) and the precision of alignments by three algorithms are illustrated in Figs. 11, 12, and 13.

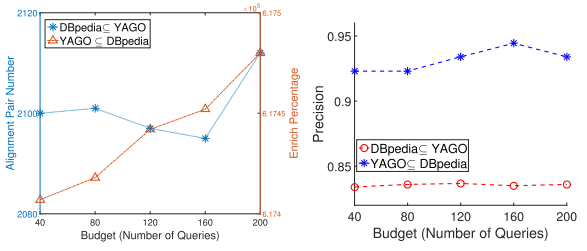
For the alignment numbers, shown in Figs. 11a, 12a, 13a, the alignment number of DBpedia $\sqsubseteq$ YAGO and YAGO $\sqsubseteq$ DBpedia pairs could be increased or decreased, not linearly related to the query number (budget). Although we prune some pairs according to the answers, we would add some new pairs in some cases as well . Take the query with "dbp: infrastructure $\sqsubseteq$ y:facility" and the contextual information of "y:facility", we have "y:gas_system" and"y:water_system" in its children list and both classes are the children of query "infrastructure". Therefore, the DBpedia *generalization/specification* pairs of YAGO $\sqsubseteq$ are added. As shown in Figs. 11b, 12b, and 13b, for different query selection strategies with varying budget, the precision of alignment results do not change much. The precision of the alignment for DBpedia$\subseteq$YAGO is around 85 percent and YAGO $\subseteq$ DBpedia is around 93 percent. The reason is that the input of the *generalization/specification* pairs before filtering has a good precision, with the query selection and pruning according to the answers, we improve the position of classes, which is more concrete and specific. For example, we have a "dbp: language $\sqsubseteq$ y:abstraction" pair, given to the crowd with the contextual information with "y:abstraction". After the query, we get "dbp:language $\sqsubseteq$ y:communication", where "communication" is a child of "abstraction". We improve the position of class "language", but the alignment pairs

4. http://webdam.inria.fr/paris/



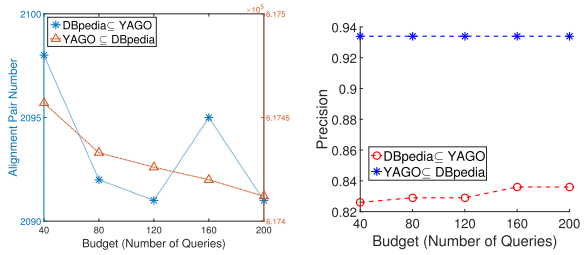
(a) Class Alignment Pairs (b) Precision of Alignment

Fig. 11. Class alignment result of random selection.



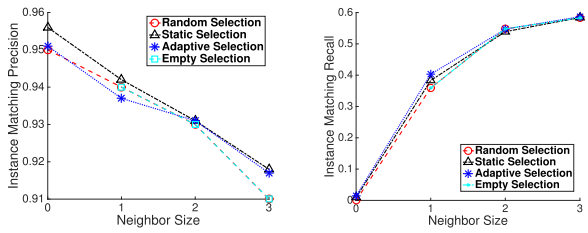
(a) Class Alignment Pairs (b) Precision of Alignment

Fig. 12. Class alignment result of greedy selection.



(a) Class Alignment Pairs (b) Precision of Alignment

Fig. 13. Class alignment result of adaptive selection.



(a) Precision (b) Recall

Fig. 14. Instance matching performance.

"dbp:language ⊑ y:abstraction" and "dbp:language ⊑ y:communication" are both correct. Therefore, combine the precision result with PFN, PIN result, we find that with our crowdsourced class alignment, we achieve accurate and precise alignment results (with comparative precision compared to Paris but improve the "in-concise" alignment pairs).

6.4 Instance Matching

We adopt the s-hop blocking-based instance matching approach, which would reduce the computing complexity to $O(N_1 * M)$, where M is the average candidate entity size of the entities from KB1. We vary the size of s and compare the precision and recall of instance matching result.

Both DBpedia and YAGO have millions of instances, if we direct compute the entities of s-hop neighbor classes, the average number is 210,123, which is still not efficient enough. Therefore, to optimize the computation, we first use the strategy in [8] to reduce the complexity to $O(N_1 * p^2)$, where p is the average statements that an entity occurs; then we use the post-filtering strategy, for each pair (e_1, e_2), we compute the parent class sets C_1, C_2 and check whether there exists $c_i \in C_1$ and $c_j \in C_2$ that c_i and c_j are within the distance of s. The average number of $|C_1| * |C_2|$ is 20. We evaluate the precision and recall of the instance matching algorithm based on various class alignment results by different local tree query selection strategies (under budget 200, 10 percent of alignment pair number): Random selection, static selection, adaptive selection and empty selection (where no selection operation performed on the initial PARIS class alignment result). The evaluation result is shown in Fig. 14. We vary s from $0 \sim 3$, note that the Empty Selection have no result when s equals to 0 as there is no equivalent relationship in PARIS class alignment result. We can observe that with larger neighbor size, the precision decreases from 95 to 91 percent and the recall increases from 38 to 58 percent. Depends on different applications, we can adopt different neighbor size, for high precision and accuracy, a smaller neighbor size is preferred and for high recall requirement, larger neighbor size should be adopted. Considering different query selection strategies, we can observe that overall the adaptive selection strategy and static selection strategy outperform the random selection and empty selection strategy with various neighbor sizes. Considering the F1 score, the adaptive strategy achieves the best performance, followed by static selection strategy, which validates the effectiveness of proposed taxonomy integration approach.

7 Related Work

7.1 Knowledge Base Integration

The problem of knowledge base (ontology) integration has been conducted by a large amount of research works, as ontology integration is an important technique to the semantic web. Aumueller et al. [29] demonstrates an ontology matching tool COMA++, which combines different matching metrics, i.e., similarity measure, linearly combining lexical, structural and extensional similarities, to support ontology matching. Udrea et al. proposes an ILIADS algorithm [10] to integrate the data matching and logical reasoning techniques to conduct ontology integration. Jean-Mary et al. [30] integrates two ontologies using lexical and structural features and calculates a similarity measure to derive the alignment, and then verified it to avoid semantic inconsistencies. PARIS [8], proposed by Suchanek et al., presents an automatic approach for ontology alignment, in which not only instances but also classes and relations are aligned. Julien et al. [9] demonstrates a more efficient algorithm, SiGMa, for large-scale knowledge base alignment. SiGMa adopted an iterative propagation algorithm to align the instances between KBs. Please refer to the [31], [32], [33] for the survey of ontology matching and knowledge base management related issues. Most of the previous works only consider the equivalence or subclass relationship between classes and cannot handle both of them as our work does.

Some work try to include user involvement into ontology integration process [22], [23], [34]. These works either limit user involvement to validating candidate subclass/superclass

mappings, and do not allow users to identify *semantic relationship* among classes or can not be applied to large scale knowledge base (ontology). The most similar work to ours is [27], in which an active learning framework is proposed for ontology matching. However, they only considers small number of queries and neglects *semantic* relationship between classes. For instance matching, the *blocking-based* techniques have long been adopted, i.e., [17], [35]. In our work, we do not work on the effectiveness of how to construct the block. We adopt the naive blocking strategy based on the class integration results and verify the efficiency and effectiveness of the instance matching.

7.2 Crowdsourced Data Management

Human computation has been used for centuries, which describes computation performed by a human and human computation systems organize human efforts to carry out computation [36]. With the popularity of commercial platforms such as Amazon MTurk (AMT) and CrowdFlower, the source of human is broadened to the crowd, namely, crowdsourcing. Recently, the increasing popularity of crowdsourcing brings new trends to adopt online crowd as a Human Processing Unit (HPU) and leverage the power of the crowd to tackle human intrinsic tasks, such as entity resolution [15], [16], [37], table matching [37], schema matching [38], information acquisition [39], taxonomy construction [40], [41], ontology integration [27], data cleaning [42] and so forth. Many tasks that cannot be addressed by machines perfectly have been solved by the crowdsourcing.

Moreover, several recent work have been conducted to improve the performance of crowdsourcing systems. Zheng et al. propose a quality-aware task assignment system to optimize the online task assignment [43]; Fan et al. propose a framework to adaptively assign microtasks to appropriate workers [44]. These works treat workers differently in the task assignment process which is an orthogonal issue as we regard workers equally.

8 Conclusion

In this paper, we propose a framework for large scale knowledge base integration through human intelligence. The core problem in KB integration, i.e., taxonomy integration, is formulated as the *Local Tree Based Query Selection* problem, which is proved to be NP-hard. To solve this optimization problem, we propose two greedy-based query selection algorithms, i.e., static query selection algorithm and adaptive query selection algorithm. Based on the taxonomy integration result, we align the instance through a s-hop neighbor based blocking strategy. Finally, we verify our proposed algorithms through extensive experimental studies on real large scale KBs. In particular, experimental results demonstrate that our approach can operate the KB integration task both efficiently and effectively.

Appendix

A. Proof of Lemma 1

Proof. Given a pair of nodes s and t, there are three *semantic* relationships:

First, if $s \sqsupseteq t$, then we know that node s is the ancestor of node t. Then, the node s with each node in the ancestor set of node s, i.e., $ans(s)$, must also be the ancestor of each descendant of node t ($des(t)$). Therefore, each candidate pair (p, q) with $p \in ans(s)$ and $q \in des(t)$, the subclass relationship does not hold and should be pruned. The number of pairs to be pruned is $\{(|ans(s)| + 1) \cdot |des(t)|\}$.

Second, if we know $s \sqsubseteq t$, then node s is the descendant of node t. In this case, each node in the descendant set of s, $des(s)$ is also a descendant of node t. Therefore, those nodes which are not descendant of t are filtered out and corresponding candidate pairs are pruned. The number of pairs to be pruned is $\{|des(s)| \cdot (M - |\, des(t)|)\}$.

If the relationship is *others*, $s \perp t$, which means that s is neither an ancestor nor a descendant of t, then all the candidate pairs (s, t'), where $t' \in des(t)$, are pruned. The number of pairs to be pruned is $\{|des(t)|\}$. □

B. Proof of Lemma 2

Proof. We prove the equivalence of Equations (8) and (9) for computing the aggregate utility of a query set $\mathcal{Q}$. We expand the formula in Equation (8) as follow:

$$\begin{aligned} U(\mathcal{Q}) &= \sum_{cp_{ij} \in CP} \sum_{R_p \in \mathcal{A}} Pr(R_p) \cdot Pru(cp_{ij}|R_p) \\ &= \sum_{cp_{ij} \in CP} \sum_{R_p \in \mathcal{A}} \prod_{a_{st} \in R_p} Pr(a_{st}) \left(1 - \prod_{a_{st} \in R_p} (1 - Pru(cp_{ij}|a_{st}))\right) \\ &= \sum_{cp_{ij} \in CP} 1 - \sum_{R_p \in \mathcal{A}} \prod_{a_{st} \in R_p} Pr(a_{st})(1 - Pru(cp_{ij}|a_{st})). \end{aligned} \tag{15}$$

Next, we analyze the expected pruning power of a query $\mathcal{Q}$, of size k, on a single candidate pair cp_{ij}. Since the possible answer results are 4^k, $\mathcal{A} = \{A(Q_1) \times A(Q_2) \times \cdots \times A(Q_k)\}$, where $A(Q_s) = \{a_{s1}, a_{s2}, a_{s3}, a_{s4}\}$ is the answer set of query Q_s. Therefore, we have

$$\begin{aligned} &Pru(cp_{ij}|\mathcal{Q}) \\ &= 1 - [Pr(a_{11})(1 - Pru(cp_{ij}|a_{11}))Pr(a_{21})(1 - Pru(cp_{ij}|a_{21})) \\ &\quad \cdots Pr(a_{k1})(1 - Pru(cp_{ij}|a_{k1})) + Pr(a_{11})(1 - Pru(cp_{ij}|a_{11})) \\ &\quad Pr(a_{21})(1 - Pru(cp_{ij}|a_{21})) \cdots Pr(a_{k2})(1 - Pru(cp_{ij}|a_{k2})) + \\ &\quad \cdots + Pr(a_{14})(1 - Pru(cp_{ij}|a_{14}))Pr(a_{24})(1 - Pru(cp_{ij}|a_{24})) \\ &\quad \cdots Pr(a_{k4})(1 - Pru(cp_{ij}|a_{k4}))] \\ &= 1 - \sum_{t=1}^{4} Pr(a_{1t})(1 - Pru(cp_{ij}|a_{1t}))[Pr(a_{21})(1 - Pru(cp_{ij}|a_{21})) \\ &\quad \cdots Pr(a_{k1})(1 - Pru(cp_{ij}|a_{k1})) + \cdots + Pr(a_{24})Pru(cp_{ij}|a_{24}) \\ &\quad \cdots Pr(a_{k4})(1 - Pru(cp_{ij}|a_{k4}))] \\ &= 1 - \left(\sum_{t=1}^{4} Pr(a_{1t})(1 - Pru(cp_{ij}|a_{1t}))\right)[Pr(a_{21})(1 - Pru(cp_{ij}|a_{21})) \\ &\quad \cdots Pr(a_{k1})(1 - Pru(cp_{ij}|a_{k1})) + \cdots + Pr(a_{24})Pru(cp_{ij}|a_{24}) \\ &\quad \cdots Pr(a_{k4})(1 - Pru(cp_{ij})|a_{k4})] \\ &= 1 - (1 - Pru(cp_{ij}|Q_1))[Pr(a_{21})(1 - Pru(cp_{ij}|a_{21})) \cdots \\ &\quad Pr(a_{k1})(1 - Pru(cp_{ij}|a_{k1})) + \cdots + Pr(a_{24})Pru(cp_{ij}|a_{24}) \cdots \\ &\quad Pr(a_{k4})(1 - Pru(cp_{ij})|a_{k4})] \\ &\quad \vdots \\ &= 1 - (1 - Pru(cp_{ij}|Q_1)(1 - Pru(cp_{ij}|Q_2)) \cdots (1 - Pru(cp_{ij}|Q_k)) \\ &= 1 - \prod_{Q_i \in \mathcal{Q}} (1 - Pru(cp_{ij}|Q_i)). \end{aligned} \tag{16}$$

Therefore, based on Equations (16) and (15), we have

$$U(\mathcal{Q}) = \sum_{cp_{ij}\in CP}\sum_{R_p\in\mathcal{A}} Pr(R_p)\cdot Pru(cp_{ij}|R_p) = \sum_{cp_{ij}\in CP} 1-\prod_{Q_i\in\mathcal{Q}}(1-Pru(cp_{ij}|Q_i)). \quad (17)$$

□

C. Proof of Theorem 3

Proof. First, we compute the pruning power on a single pair cp_{ij} from $\{R_p^1 \cup a_t\}$, where $R_p \in \mathcal{A}_1$ and $a_t \in Q_t$.

$$\begin{aligned} &Pru(cp_{ij}|R_p^1\cup a_t)\\ &= 1-\prod_{a\in R_p^1\cup a_t}(1-Pru(cp_{ij}|a))\\ &= 1-(1-Pru(cp_{ij}|a_t))\prod_{a_s\in R_p^1}(1-Pru(cp_{ij}|a_s))\\ &= 1-\prod_{a_s\in R_p^1}(1-Pru(cp_{ij}|a_s))+Pru(cp_{ij}|a_t)\prod_{a_s\in R_p^1}(1-Pru(cp_{ij}|a_s))\\ &= Pru(cp_{ij}|R_p^1)+Pru(cp_{ij}|a_t)\cdot(1-Pru(cp_{ij}|R_p^1)). \end{aligned} \quad (18)$$

According to Equation (18), we have the increased pruning power on cp_{ij} by adding a query result of a_t to R_p^1

$$\begin{aligned} &\Delta(cp_{ij}|R_p^1,a_t)\\ &= Pru(cp_{ij}|R_p^1\cup a_t)-Pru(cp_{ij}|R_p^1)\\ &= Pru(cp_{ij}|a_t)\cdot(1-Pru(cp_{ij}|R_p^1))). \end{aligned} \quad (19)$$

Based on Equation (19), we have $\Delta(cp_{ij}|R_p^1,a_t)\geq\Delta(cp_{ij}|R_q^2,a_t)$, if $R_p^1\subset R_q^2$. Next, we compute the increase of utility by adding an answer set $\mathcal{A}_1$ of Q_t

$$\begin{aligned} &Pru(cp_{ij}|\mathcal{A}_1\cup A)\\ &= \sum_{(R_p^1\cup a_t)\in\mathcal{A}_1\cup A} Pr(R_p^1\cup a_t)Pru(cp_{ij}|R_p^1\cup a_t)\\ &= \sum_{R_p^1\in\mathcal{A}_1}\sum_{a_t\in A} Pr(R_p^1)Pr(a_t)Pru(cp_{ij}|R_p^1\cup a_t)\\ &= \sum_{R_p^1\in\mathcal{A}_1} Pr(R_p^1)\sum_{a_t\in A}Pr(a_t)(Pru(cp_{ij}|R_p^1)+Pru(cp_{ij}|a_t)\\ &\quad (1-Pru(cp_{ij}|R_p^1)))\\ &= Pr(cp_{ij}|\mathcal{A}_1)+Pru(cp_{ij}|A)-Pru(cp_{ij}|\mathcal{A})Pru(cp_{ij}|A). \end{aligned} \quad (20)$$

Therefore, we have the delta of utility of adding a query Q with answer set is A is

$$\Delta(cp_{ij}|\mathcal{A}_1,A) = Pru(cp_{ij}|A)-Pru(cp_{ij}|\mathcal{A})Pru(cp_{ij}|A). \quad (21)$$

And the delta utility of adding a query Q_t to query set $\mathcal{Q}$ is

$$\begin{aligned} \Delta U(Q) &= U(\mathcal{Q}_1\cup Q_t)-U(\mathcal{Q}_1)\\ &= \sum_{cp_{ij}\in CP} U(cp_{ij}|\mathcal{A}_1\cup A)-U(cp_{ij}|\mathcal{A}_1)\\ &= \sum_{cp_{ij}\in CP} Pru(cp_{ij}|A)-Pru(cp_{ij}|\mathcal{A}_1)Pru(cp_{ij}|A)\\ &= \sum_{cp_{ij}\in CP} Pru(cp_{ij}|A)(1-Pru(cp_{ij}|\mathcal{A}_1)). \end{aligned} \quad (22)$$

Similarly, delta utility of adding query Q_t to query set $\mathcal{Q}_1$ is

$$U(\mathcal{Q}_2\cup Q_t)-U(\mathcal{Q}_2) = \sum_{cp_{ij}\in CP} Pru(cp_{ij}|A)(1-Pru(cp_{ij}|\mathcal{A}_2)). \quad (23)$$

As we know $Pru(cp_{ij}|\mathcal{A}_2)\geq Pru(cp_{ij}|\mathcal{A}_1)$, combined with Equations (22) and (23), we have

$$U(\mathcal{Q}_1\cup Q_t)-U(\mathcal{Q}_1)\geq U(\mathcal{Q}_2\cup Q_t)-U(\mathcal{Q}_2). \quad (24)$$

□

ACKNOWLEDGMENTS

The authors are grateful to anonymous reviewers for their constructive comments on this work. The work is partially supported by the Hong Kong RGC Project N HKUST637/13, National Grand Fundamental Research 973 Program of China under Grant 2014CB340303, the National Science Foundation of China (NSFC) under Grant No. 61502021, 61328202, and 61532004, NSFC Guang Dong Grant No. U1301253, and Microsoft Research Asia Fellowship 2012. The Fostering Project of Dominant Discipline and Talent Team of Shandong Province Higher Education Institutions.

REFERENCES

[1] J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum, "YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia," *Artif. Intell.*, vol. 194, pp. 28–61, 2013.

[2] W. Wu, H. Li, H. Wang, and K. Q. Zhu, "Probase: A probabilistic taxonomy for text understanding," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2012, pp. 481–492.

[3] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, "Freebase: A collaboratively created graph database for structuring human knowledge," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2008, pp. 1247–1250.

[4] O. Etzioni, et al., "Web-scale information extraction in KnowitAll: (preliminary results)," in *Proc. 13th Int. Conf. World Wide Web*, 2004, pp. 100–110.

[5] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives, "DBpedia: A nucleus for a Web of open data," in *Proc. 6th Int. Semantic Web 2nd Asian Conf. Asian Semantic Web Conf.*, 2007, pp. 722–735.

[6] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. H. Jr., and T. M. Mitchell, "Toward an architecture for never-ending language learning," in *Proc. 24th AAAI Conf. Artif. Intell.*, 2010, pp. 1306–1313.

[7] F. Niu, C. Zhang, C. Re, and J. W. Shavlik, "DeepDive: Web-scale knowledge-base construction using statistical learning and inference," in *Proc. 2nd Int. Workshop Searching Integr. New Web Data Sources*, 2012, pp. 25–28.

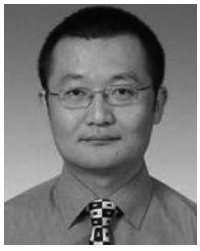
[8] F. M. Suchanek, S. Abiteboul, and P. Senellart, "PARIS: Probabilistic alignment of relations, instances, and schema," *Proc. VLDB Endowment*, vol. 5, pp. 157–168, 2011.

[9] S. Lacoste-Julien, K. Palla, A. Davies, G. Kasneci, T. Graepel, and Z. Ghahramani, "SIGMa: Simple greedy matching for aligning large knowledge bases," in *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2013, pp. 572–580.

[10] O. Udrea, L. Getoor, and R. J. Miller, "Leveraging data and structure in ontology integration," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2007, pp. 449–460.

[11] J. Euzenat and P. Shvaiko, *Ontology Matching*. Berlin, Germany: Springer, 2007.
[12] P. Giaretta and N. Guarino, "Ontologies and knowledge bases towards a terminological clarification," in *Towards Very Large Knowledge Bases: Knowledge Building & Knowledge Sharing*. Amsterdam, The Netherlands: IOS Press 1995.
[13] A. Nandi and P. A. Bernstein, "HAMSTER: Using search clicklogs for schema and taxonomy matching," *Proc. VLDB Endowment*, vol. 2, pp. 181–192, 2009.
[14] P. Papadimitriou, P. Tsaparas, A. Fuxman, and L. Getoor, "TACI: Taxonomy-aware catalog integration," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 7, pp. 1643–1655, Jul. 2013.
[15] J. Wang, T. Kraska, M. J. Franklin, and J. Feng, "CrowdER: Crowdsourcing entity resolution," *Proc. VLDB Endowment*, vol. 5, pp. 1483–1494, 2012.
[16] N. Vesdapunt, K. Bellare, and N. N. Dalvi, "Crowdsourcing algorithms for entity resolution," *Proc. VLDB Endowment*, vol. 7, pp. 1071–1082, 2014.
[17] G. Papadakis, E. Ioannou, T. Palpanas, C. Niederée, and W. Nejdl, "A blocking framework for entity resolution in highly heterogeneous information spaces," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 12, pp. 2665–2682, Dec. 2013.
[18] J. J. Jung, "Taxonomy alignment for interoperability between heterogeneous virtual organizations," *Expert Syst. Appl.*, vol. 34, pp. 2721–2731, 2008.
[19] R. Meng, Y. Tong, L. Chen, and C. C. Cao, "CrowdTC: Crowdsourced taxonomy construction," in *Proc. IEEE Int. Conf. Data Mining*, 2015, pp. 913–918.
[20] E. D. Simpson, et al., "Language understanding in the wild: Combining crowdsourcing and machine learning," in *Proc. 24th Int. Conf. World Wide Web*, 2015, pp. 992–1002.
[21] V. Crescenzi, P. Merialdo, and D. Qiu, "A framework for learning web wrappers from the crowd," in *Proc. 22nd Int. Conf. World Wide Web*, 2013, pp. 261–272.
[22] I. F. Cruz, C. Stroe, and M. Palmonari, "Interactive user feedback in ontology matching using signature vectors," in *Proc. IEEE 28th Int. Conf. Data Eng.*, 2012, pp. 1321–1324.
[23] I. F. Cruz, F. Loprete, M. Palmonari, C. Stroe, and A. Taheri, "Pay-as-you-go multi-user feedback model for ontology matching," in *Proc. 19th Int. Conf. Knowl. Eng. Knowl. Manage.*, 2014, pp. 80–96.
[24] A. Parameswaran, A. D. Sarma, H. Garcia-Molina, N. Polyzotis, and J. Widom, "Human-assisted graph search: It's okay to ask questions," *Proc. VLDB Endowment*, vol. 4, pp. 267–278, 2011.
[25] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng, "Leveraging transitive relations for crowdsourced joins," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013, pp. 229–240.
[26] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions," *Math. Program.*, vol. 14, pp. 265–294, 1978.
[27] F. Shi, J. Li, J. Tang, G. T. Xie, and H. Li, "Actively learning ontology matching via user interaction," in *Proc. 8th Int. Semantic Web Conf. Semantic Web*, 2009, pp. 585–600.
[28] Z. Chen, et al. "gMission: A general spatial crowdsourcing platform," *Proc. VLDB Endowment*, vol. 7, pp. 1629–1632, 2014.
[29] D. Aumueller, H. H. Do, S. Massmann, and E. Rahm, "Schema and ontology matching with COMA++," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2005, pp. 906–908.
[30] Y. R. Jean-Mary, E. P. Shironoshita, and M. R. Kabuka, "Ontology matching with semantic verification," *J. Web Semantic*, vol. 7, pp. 235–251, 2009.
[31] P. Shvaiko and J. Euzenat, "Ontology matching: State of the art and future challenges," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 1, pp. 158–176, Jan. 2013.
[32] L. Otero-Cerdeira, F. J. Rodríguez-Martínez, and A. Gómez-Rodríguez, "Ontology matching: A literature review," *Expert Syst. Appl.*, vol. 42, pp. 949–971, 2015.
[33] M. L. Brodie and J. Mylopoulos, *On Knowledge base Management Systems: Integrating Artificial Intelligence and Database Technologies*. Berlin, Germany: Springer, 2012.
[34] C. Sarasua, E. Simperl, and N. F. Noy, "CrowdMap: Crowdsourcing ontology alignment with microtasks," in *Proc. 11th Int. Conf. Semantic Web*, 2012, pp. 525–541.
[35] G. Papadakis, E. Ioannou, C. Niederée, and P. Fankhauser, "Efficient entity resolution for large heterogeneous information spaces," in *Proc. 4th ACM Int. Conf. Web Search Data Mining*, 2011, pp. 535–544.
[36] M. Lease and O. Alonso, "Crowdsourcing and human computation, introduction," in *Proc. Encyclopedia Social Netw. Anal. Mining*, 2014, pp. 305–315.
[37] J. Fan, M. Lu, B. C. Ooi, W. Tan, and M. Zhang, "A hybrid machine-crowdsourcing system for matching Web tables," in *Proc. IEEE 30th Int. Conf. Data Eng.*, 2014, pp. 976–987.
[38] C. J. Zhang, L. Chen, H. V. Jagadish, and C. C. Cao, "Reducing uncertainty of schema matching via crowdsourcing," *Proc. VLDB Endowment*, vol. 6, pp. 757–768, 2013.
[39] S. K. Kondreddi, P. Triantafillou, and G. Weikum, "Combining information extraction and human computing for crowdsourced knowledge acquisition," in *Proc. IEEE 30th Int. Conf. Data Eng.*, 2014, pp. 988–999.
[40] L. B. Chilton, G. Little, D. Edge, D. S. Weld, and J. A. Landay, "Cascade: Crowdsourcing taxonomy creation," in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, 2013, pp. 1999–2008.
[41] J. Bragg, Mausam, and D. S. Weld, "Crowdsourcing multi-label classification for taxonomy creation," in *Proc. 1st AAAI Conf. Human Comput. Crowdsourcing*, 2013, pp. 25–33.
[42] X. Chu, et al., "KATARA: A data cleaning system powered by knowledge bases and crowdsourcing," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp 1247–1261.
[43] Y. Zheng, J. Wang, G. Li, R. Cheng, and J. Feng, "QASCA: A quality-aware task assignment system for crowdsourcing applications," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 1031–1046.
[44] J. Fan, G. Li, B. C. Ooi, K. Tan, and J. Feng, "iCrowd: An Adaptive crowdsourcing framework," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 1015–1030.

Rui Meng received the BS degree in software engineering from Shandong University, China, in 2013. She is currently working toward the PhD degree in the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. Her research interests include crowdsourcing and knowledge management.



Lei Chen received the PhD degree in computer science from the University of Waterloo, Canada, in 2005. He is currently a professor in the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. His research interests include crowdsourcing over social media, social media analysis, probabilistic and uncertain databases, and privacy-preserved data publishing. He is a member of the IEEE.

Yongxin Tong received the PhD degree in computer science and engineering from the Hong Kong University of Science and Technology (HKUST), in 2014. He is currently an associate professor in the School of Computer Science and Engineering, Beihang University, China. Before that, he served as a research assistant professor and a postdoc fellow with HKUST. His research interests include crowdsourcing, uncertain data mining and management, and social network analysis.

Chen Zhang received the PhD degree from the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, in 2015. He is currently a postdoctoral research fellow with the Hong Kong University of Science and Technology as well as an associate professor at the Shandong University of Finance and Economics. His research interests include crowdsourcing and data integration.