Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence

ARIA: Asymmetry Resistant Instance Alignment

Sanghoon Lee and **Seung-won Hwang**
Pohang University of Science and Technology (POSTECH), Korea, Republic of
{sanghoon, swhwang}@postech.edu

Abstract

We study the problem of instance alignment between knowledge bases (KBs). Existing approaches, exploiting the "symmetry" of structure and information across KBs, suffer in the presence of asymmetry, which is frequent as KBs are independently built. Specifically, we observe three types of asymmetries (in concepts, in features, and in structures). Our goal is to identify key techniques to reduce accuracy loss caused by each type of asymmetry, then design Asymmetry-Resistant Instance Alignment framework (ARIA). ARIA uses two-phased blocking methods considering concept and feature asymmetries, with a novel similarity measure overcoming structure asymmetry. Compared to a state-of-the-art method, ARIA increased precision by 19% and recall by 2%, and decreased processing time by more than 80% in matching large-scale real-life KBs.

Introduction

Public knowledge bases (KBs), e.g., DBpedia (Lehmann et al. 2014) and YAGO (Biega, Kuzey, and Suchanek 2013) have become abundant. Linking entities within KBs would increase their utility as information sources, but this process is difficult because they have been assembled independently and therefore may describe entities in different terms. State-of-the-art KB-linking approaches such as PARIS (Suchanek, Abiteboul, and Senellart 2011), ObjCoref (Hu, Chen, and Qu 2011), and SiGMa (Lacoste-Julien et al. 2012) exploit the symmetries of relations, concepts and instances in two KBs. However, the accuracy of these approaches decreases in the presence of the following asymmetries.

- **Concept asymmetry**: KBs may represent concepts in different ways (e.g., `author` = `writer`).
- **Feature asymmetry**: Features associated with an entity may have differing discriminative power (e.g., `firstName` can be effective for distinguishing entities, but a common name like "Steve" or `occupation` is not). Even if the features are low discriminative, a combination of such features would be highly selective to distinguish between entities.

- **Structure asymmetry**: The same entity may have fewer features (e.g., name, company) in one KB than in another (e.g., name, company, residence, spouse).

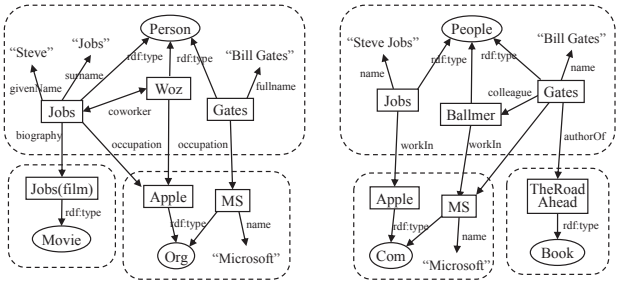
To address the challenges posed by these asymmetries, we use three approaches.

First, we compute concept similarities to overcome concept asymmetry. Although concept is a critical clue for "blocking", of grouping matching candidates into a block, asymmetry affects the robustness of such process. For example, `author` and `writer` have different names, but share common instances, which strongly supports the possibility of two being the same concept. Once we identify such concept pairs (c, c'), we can significantly reduce search space by blocking instances in c and c' as a group of potential matches.

Second, we use feature combination for blocking to overcome feature asymmetry. Most existing blocking (or clustering) methods use a single criterion to assemble groups of similar instances (Baxter, Christen, and Churches 2003). For example, `firstName` can be used as a criterion to block instances with the same name, which would result in a coarse block of "Steve". For such blocks, we can use another cycle of *inner-group blocking* to further divide them into sub-blocks by `occupation` to generate a more reasonable sized block of both "Steve" and `Apple`.

Lastly, we propose a robust instance similarity measure to overcome structure asymmetry. This measure is used to rank possible matches within the block by similarity scores. Past approaches aggregated features, e.g., into a virtual document (Qu, Hu, and Cheng 2006), then compared them as points in a vector space. In this approach, if one entity has more information in one KB than in another, the similarity score is underestimated because the additional information in one KB cannot be found in the other and is counted as a non-match. Instead, we propose triple similarity to consider only the matching information in similarity computation.

The key contributions of this paper are identifying techniques to overcome three major types of asymmetries and building it into a new alignment called Asymmetry Resistant Instance Alignment (ARIA), that can align corresponding instances in KBs despite the existence of these asymmetries.



(a) KB_X (b) KB_Y

Figure 1: Two toy example KBs. The rectangle and circle nodes represent instances and concepts, respectively. Prefixes of resources are omitted such as x: for resources in (a) KB_X and y: for resources in (b) KB_Y.

Preliminaries

This section introduces fundamental notions of KB, and formally defines the instance alignment problem.

Knowledge bases (KBs)

We use "knowledge base" to mean a collection of knowledge with concepts, instances, literals, and relations among them. We formally define the KB as a tuple $KB = \langle C, I, L, R, T \rangle$ where C is a set of concepts, I is a set of instances, L is a set of literals, R is a set of relations, and T is a set of triples over $I \times R \times (C \cup I \cup L)$.

We use the triple notion of the Resource Description Framework (RDF) to present facts in KBs. An RDF triple consists of three components, $\langle subject, predicate, object \rangle$. The *subject* is an instance, the *object* is a concept, instance, or literal, and the *predicate* is a relation between subject and object. The object describes information about predicate for subject instance. Every element in C, I, and R is represented by a uniform resource identifier (URI), like denoting http://dbpedia.org/resource/Steve_Jobs, briefly as d:Steve_Jobs where prefix d: means http://dbpedia.org/resource/. For simplicity, we will omit the prefix when there is no ambiguity.

The triples can be presented in a graph model (Figure 1). The subjects and objects are nodes in the graph, and triples are edges with labels of predicates. For example, KB_X contains a triple $\langle$`Jobs`, `rdf:type`, `Person`$\rangle$ which describes instance `Jobs` belongs to `Person` domain, and presented as an edge from `Jobs` node to `Person` with the label `rdf:type`.

Instance alignment problem

An entity is instantiated as a unique resource in a KB, which implies that it includes no duplicate instances for the same entity in the KB. However, because KBs are constructed by different repositories, two instances in different KBs may refer to the same entity.

Our goal is to link instances that represent the same entities across KBs. We formally state the instance alignment problem as defined by Euzenat and Shvaiko.

Definition 1 (Instance alignment problem) *The instance alignment is to find a function f which a pair of KBs to match (KB_X and KB_Y), an input alignment (A), a set of parameters (p), a set of oracles and resources (r), returns an alignment (A') between these KBs:*

$$A' = f(KB_X, KB_Y, A, p, r)$$

where the alignment (A or A') consists of a set of instance pairs.

The alignment function takes input alignment (A) which can be seeds of matching process, and we are trying to extend the alignment to cover as many as instances in KB_X and KB_Y. As external knowledge (r), we can exploit OWL semantics such as `owl:sameAs` or `owl:InverseFunctionalProperty` which is further discussed in the next section. The output (A') consists of pairs of instances ($\langle x, y \rangle$), one for each KB. Besides the instance alignment, we also utilize relation and concept alignment (also known as ontology matching) which can be inferred from the instance alignment.

We assume that the output of instance alignment is constrained to be one-to-one mapping, because alignment with the constraint is reportedly more accurate than without such constraint (Gemmell, Rubinstein, and Chandra 2011).

Concept-based Blocking

The first step of our proposed approach is to restrict candidates that belong to the equivalent concept. After identifying equivalent concepts between KBs, we can reduce the size of alignment problem to smaller domains. In the Figure 1, if two concepts `x:Person` and `y:People` are given as an equivalent concept pair, we can restrict matching candidates of `x:Jobs` to belonging to `y:People`, such as `y:Jobs` and `y:Ballmer`.

The challenge in concept-based blocking is that the equivalent concepts in two KBs can be represented by different names. Therefore, we conduct concept matching based on instances. We exploit seed matches to estimate concept similarities, obtained as follows:

- **P1** (Prior knowledge of `owl:sameAs`[1]): This links the same entity across different KBs. We can directly take an instance pair with `owl:sameAs` relation as a match.
- **P2** (Prior knowledge of `owl:InverseFunctionalProperty`): The inverse functional property (IFP) uniquely determines its subject instance with respect to the object value. Two different instances but having the same object for the IFP can be inferred to be the same entity. For example, given two triples $\langle$`Jobs`, `tel`, "555-1996"$\rangle$ and $\langle$`Jobs_2`, `tel`, "555-1996"$\rangle$ in a KB and relation `tel` is declared to be inverse functional, then instances `Jobs` and `Jobs_2` refer to the same entity. With this property, we regard two instances in different KBs of same literal for IFPs as a seed.
- **E** (Extracted seeds): In the absence of prior knowledge, such as **P1** and **P2**, we can infer the IFP heuristically from a unique object value which describes only one instance in a KB. We use the unique object values to approximate

[1] owl: refers to http://www.w3.org/2002/07/owl#

Table 1: Contingency table of two concept variables, where $n_{00}, n_{01}, n_{10}, n_{11}$ are the number of seeds for corresponding events.

	$c_Y = 0$	$c_Y = 1$	total
$c_X = 0$	n_{00}	n_{01}	n_{0*}
$c_X = 1$	n_{10}	n_{11}	n_{1*}
total	n_{*0}	n_{*1}	n

IFP, and find instance pairs of two KBs which have the same values of IFP. For example, for given two triples ⟨`x:Jobs`, `x:tel`, "555-1995"⟩ and ⟨`y:Jobs`, `y:phone`, "555-1995"⟩ in different KBs, we consider the two subject instances are equivalent, when no other triples of ⟨∗, ∗,"555-1995"⟩ occur in the KBs, even if the relations `x:tel` and `y:phone` are not defined as IFP.

Our framework does not require any prior knowledge such as **P1** and **P2**, although they can be used if available. To make this point, we use only the third seed extraction method (**E**) because `owl:sameAs` or IFP may not be given for some KBs. We formally define a set of seed matches as follows.

Definition 2 (Seed matches) *For all literals v in two KB literal sets (L_X, L_Y) and triple sets (T_X, T_Y), if each KB contains only one instance related to the value, a set of such instance pairs is seed matches (S):*

$$S = \left\{ \langle x, y \rangle \;\middle|\; \begin{array}{c} \forall v \in L_X \cap L_Y, |\{x | \langle x, *, v \rangle \in T_X\}| = 1, \\ |\{y | \langle y, *, v \rangle \in T_Y\}| = 1 \end{array} \right\}.$$

If there are many seeds correlated to a concept pair, it suggests potential compatibility of the two concepts. For this purpose, we consider seed matches as our training data for learning the correlation between concepts. Among various statistical correlation measures, we adopt the ϕ coefficient, which is a specialized association measure of Pearson correlation for two binary variables. We map the concept c to the variable, and denote $c = 1$ for an event that contains an instance in the concept, and $c = 0$ otherwise. With this notion, we count the number of seeds for 2×2 events for two concepts (Table 1). From the contingency table of a concept pair $\langle c_X, c_Y \rangle$, the correlation ϕ is computed as

$$corr(c_X, c_Y) = \frac{n_{11}n_{00} - n_{10}n_{01}}{\sqrt{n_{1*}n_{0*}n_{*1}n_{*0}}}.$$

We stress that our goal is to match *equivalent* concepts. For example, matching `Person` with its subconcept, e.g., `Living_people`, should not be identified as a match, although the intersection of the two is significant, because such a decision may generate a needless restriction that the matching entity be currently alive. For this reason, we *conservatively* identify concept pairs whose $corr()$ is significantly high, and divide the original problem by taking input instances in the domains.

Definition 3 (Sub-instance alignment problem) *Given an instance alignment problem $f(KB_X, KB_Y, A, p, r)$, and a set of matched concept pairs A_C, a sub-instance alignment problem for a concept match $\langle c_X, c_Y \rangle \in A_C$ is*

$$A' = f(KB_X(c_X), KB_Y(c_Y), A, p, r),$$

where $KB(c) = \langle C, I(c), L, R, T \rangle$ and $I(c) = \{i | \langle i,$ `rdf:type`$, c \rangle \in T\}$.

Feature-based Blocking

The second step of our proposed approach is a finer-level blocking method within a group of candidates identified based on concepts. A challenge in this step is to overcome feature asymmetry.

This step would be trivial if two KBs share a key feature such as a social security number which uniquely defines an entity in both KBs. However, sharing a unique key defeats the whole purpose of instance alignment and thus cannot be assumed. Instead, for state-of-the-art methods such as PARIS and ObjCoref use "near-unique" literals as strong supporting evidence for blocking. However, not all entities have such discriminative literals. For example, "Steve" is a common first name and cannot distinguish Steve Jobs from Steve Ballmer, so using "Steve" as a blocking key would cluster too many instances into a group.

In contrast, using a combination of features as a blocking key can be discriminative evidence, even if each feature by itself generates coarse clusters. For example, although neither "Steve" nor `Apple` determines a small block, the combination of these features can.

More formally, we develop an algorithm (Algorithm 1) that starts with a single block of all instances. We enumerate features of instances in the block, and extract sub-blocks, one for each feature. For example, we may consider a literal feature "Steve" as blocking key at first, so that hundreds of instances are put into a block. In this level of blocking, the blocking key consists of a single feature. We use a threshold t to decide whether or not the block is sufficiently discriminative. When the block size is smaller than t, the block is accepted.

Meanwhile, because blocks by `firstName` are too coarse, we continue by extracting sub-blocks by adding key features. To reduce the number of candidates, we extract a sub-block again with another feature such as `Apple`. When there remain fewer instances related to `Apple` that are also named "Steve", those instances are put in the sub-block. This process is recursively repeated until the block is discriminative or every combination of features has been considered.

This blocking approach is related to the dynamic blocking method (McNeill, Kardes, and Borthwick 2012) in that both recursively find blocking keys by keeping the keys more constrained. However, dynamic blocking requires prior knowledge on which feature is critical for blocking, e.g., a fixed ordering of using `firstName` as the first block, then `occupation`. In contrast, our approach does not require such knowledge and can enumerate arbitrary combinations for the following reasons: First, very discriminative evidence such as an `address` may lead to low recall in some blocks, because the same address may often have different representations, e.g., "Street" and "St". We may selectively

Algorithm 1 Block(I_X, I_Y, T_X, T_Y, k, t)

Input: two instance sets (I_X, I_Y), two triple sets (T_X, T_Y), aligned concept pairs (A_C) and instance pairs (A_I), blocking key (k), and candidate degree threshold (t)
Output: a set of candidates of instance pairs (C)

```
A_L ← {⟨o, o⟩|⟨∗, ∗, o⟩ ∈ T_X} // literal features
for every feature ⟨o_1, o_2⟩ ∈ A_C ∪ A_I ∪ A_L do
  if ⟨o_1, o_2⟩ ∉ k then
    J_X ← {x ∈ I_X|⟨x, ∗, o_1⟩ ∈ T_X}
    J_Y ← {y ∈ I_Y|⟨y, ∗, o_2⟩ ∈ T_Y}
    if t < |J_X| or t < |J_Y| then
      k ← k ∪ {⟨o_1, o_2⟩}
      Block(J_X, J_Y, T_X, T_Y, k, t)
      k ← k \ {⟨o_1, o_2⟩}
    else
      C ← C ∪ (I_X × I_Y)
    end if
  end if
end for
```

use such evidence only for a block that has identical representation. Second, for some instances with a common literal, e.g., "Steve", `firstName` may not be as discriminative as in another instance. Third, enumerating all possible combinations incurs negligible cost, because after the first blocking, the number of possible second blocking keys decreases significantly, and the number of such selections is typically no more than three. Finally, it is nontrivial to predefine the fixed ordering of features for each domain of general KBs, e.g., instances of `Person` and `Location` have totally different feature sets.

Similarity Matching

In the preceding two steps, we divided instances into groups of matching candidates. Our next step is to compute scores for possible pairs in the group. Several similarity measures have been proposed in literature. Most measures focus on string similarity of literal features (Stoilos, Stamou, and Kollias 2005). Qu, Hu, and Cheng puts every feature for an instance to a document and measure the document distance. In addition to literal proximity, we should consider following two challenges.

- We must overcome structure asymmetry. For example (Figure 2), `x:Jobs` may have richer features in one KB than in the other; e.g., extension appearing only in `x:Jobs`, but this should neither discount the similarity (due to no matching extension in `y:Jobs`), nor lead to a wrong match (`y:Ballmer` with a matching literal "1956").
- We must exploit entity relationships. Having `Apple` as a neighbor with the same relation ot `Jobs` (e.g., `occupation` and `workIn`) in two KBs is a strong signal of similarity. However, as in concept-based blocking, the same resources may not identify equivalent relations.

Specifically, we identify two sub-tasks to compute instance similarities:

- **Triple selection:** We develop *triple similarity* to consider only related triples that have equivalent semantics with high similarity, and prune out the rest to avoid discounting.
- **Triple aggregation:** We define *instance similarity* which aggregates triple-level similarities for matching.

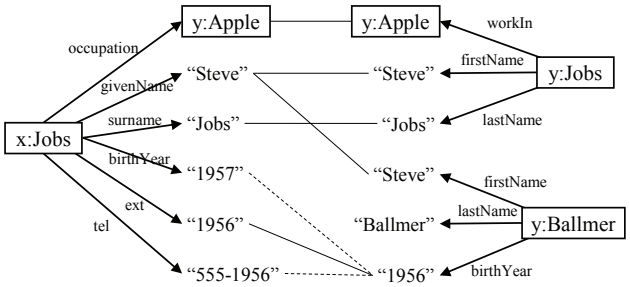


Figure 2: An example of structure symmetry. Triples with "1956" can be matched accidentally by ignoring relations.

Triple Selection

When a KB has much richer triples for an instance than another KB, this asymmetry often leads to false matching or similarity discounting. Meanwhile, matching entity pairs may score low in terms of similarity, when all triples are factored in for similarity computation.

Our goal is thus to select only the corresponding triples in two KBs. We choose triple pairs that represent the same knowledge with one-to-one constraint, whose similarities exceed a certain threshold θ. This triple matching prior to computing instance similarity can effectively eliminate information that does not appear in another instance. This measure is inspired by soft TF-IDF (Bilenko et al. 2003) which is a name similarity measure based on matching tokens. Soft TF-IDF is based on Cosine similarity with TF-IDF, but it takes token score of the most similar one for each token.

More specifically, we combine the following three types of triples in similarity computation:

- **Literal triples**: We use string similarity measures to compare literals. Various string similarity measures have been proposed for name matching, such as edit distance, Jaro-Winkler distance (Winkler 1999), and I-Sub (McNeill, Kardes, and Borthwick 2012).
- **Instance triples**: Once instance similarities have been computed, the similarity of neighboring instances can be considered as similarity flooding (Melnik, Garcia-Molina, and Rahm 2002). Seeds can be used as initial instance triple similarities.
- **Concept triples**: Concept similarity can be considered; we can reuse concept correlation computed in a concept-based blocking method, measured by instance correlation.

We stress that considering only one of these types may lead to inaccurate matching. For example (Figure 2),

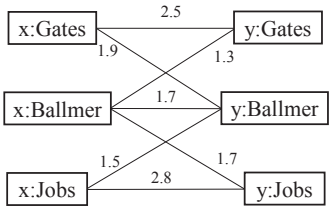


Figure 3: A bipartite graph of matching example. Edge weights represent instance similarities.

`x:Jobs` and `y:Ballmer` share the literal value "1956", which may lead to a false matching between the two. However, two triples of the literal describe entirely different facts by relations `ext` and `birthYear`. In summary, a similarity function should (1) holistically aggregate the above three types of triple, and (2) prune out asymmetric triples. Therefore, the feature similarity is defined as follows.

Definition 4 (Triple similarity) *Given two triples t_X and t_Y, the triple similarity between two triples is*

$$\begin{aligned} sim'(t_X, t_Y) &= sim'(\langle s_X, p_X, o_X\rangle, \langle s_Y, p_Y, o_Y\rangle) \\ &= corr(p_X, p_Y) \times osim(o_X, o_Y) \end{aligned}$$

where $corr()$ is a similarity by correlation of two relations, and $osim()$ is object feature similarity such as string similarity if o_X and o_Y are literals, or instance similarity if they are instances, or correlation similarity if they are concepts.

Triple Aggregation

To prune out asymmetric triples, we only consider pairs with similarity $> \theta$, which implies that these pairs have strong evidence that the instance should be aligned. In addition, unmatched triples are pruned out so that asymmetric triples do not penalize the overall similarity score of aligned instances. Then, we aggregate the survived triples holistically for instance similarity. We formally define instance similarity as follows.

Definition 5 (Instance similarity) *Given two instances x and y, instance similarity is the sum of the matched triple similarities whose values are larger than θ.*

$$\begin{aligned} sim(x, y) &= sim(T_X(x), T_Y(y)) \\ &= \sum_{t_X \in T_X(x)} \max_{\substack{t_Y \in T_Y(y), \\ sim'(t_X, t_Y) \geq \theta}} sim'(t_X, t_Y), \end{aligned}$$

where $T(x)$ is a set of triples whose subject instance is x.

After all instance similarity scores are computed for matching candidates, the final matches are selected based on the score. We model the instance alignment as a matching problem on a weighted bipartite graph, where instances are nodes connected by edges weighted by instance similarity (Figure 3). Once the relations are abstracted as a graph like this, we can align instances by adopting existing graph matching methods, which have been proposed to identify pairs that maximize the overall edge weight.

Experimental Evaluation

Settings

Evaluations were conducted on an Intel quad-core i7 3.6GHz CPU with 32 GB RAM equipped with Java 7. Alignment accuracy was measured by precision and recall. To evaluate blocking quality, we used reduction ratio (*RR*) and pair completeness (*PC*)–*RR* is the ratio of pruned instance pairs among all possible pairs, and *PC* is the ratio of true matches for all pairs. We encoded the identifiers (e.g., URIs) of instances, relations, and concepts to avoid cheating by using URI text as alignment clues.

For datasets, we used DBpedia (Lehmann et al. 2014) and YAGO (Biega, Kuzey, and Suchanek 2013), which are real-world large-scale KBs that cover millions of instances. Both datasets have links to Wikipedia webpages if instances have corresponding Wikipedia articles. Therefore, we can compose ground truth between DBpedia and YAGO by tracing the Wikipedia links; these links were used only for generating ground truth, not for alignment.

Table 2: Data statistics of two DBpedia datasets with different sizes and the YAGO dataset.

	DBpedia	DBpedia+	YAGO
#concepts	434	434	374K
#instances	2.4M	10.8M	3.0M
#relations	1.4K	49.7K	93
#triples	34.4M	136.6M	32.9M

Specifically, we use two different versions of DBPedia–namely, DBpedia and DBpedia+ (Table 2). First, DBpedia was used to compare with PARIS[2] (Suchanek, Abiteboul, and Senellart 2011) to reproduce evaluation settings in their paper. Second, we also consider a larger set DBpedia+, by merging raw infobox properties[3]. The purpose of this second dataset is to enrich the feature of one KB (to increase asymmetry), to demonstrate that the robustness of our framework is not reduced by structure asymmetry.

Alignment evaluation

We performed alignment between DBpedia and YAGO. We first found 344K seed matches by using the extracted seeds (**E**) as discussed in the Concept-based Blocking section. These seeds cover 18.8% of gold standards with near-perfect precision, based on which we compute concept correlations to identify top-3 equivalent concepts, such as person, location, and organization domains (Table 3). Other instances that do not belong to any matched concepts were clustered into an etc domain. This concept-based blocking

[2]We use PARIS as baseline, as it was empirically compared with other competitor in terms of accuracy (Lacoste-Julien et al. 2012). We adopt most recent release, i.e., of May 2013, which significantly improves (Suchanek, Abiteboul, and Senellart 2011) in terms of efficiency. PARIS is the strongest baseline we can consider as of now, to our knowledge.

[3]The snapshot of raw infobox properties is downloaded from http://wiki.dbpedia.org/Downloads39#raw-infobox-properties

enables disambiguation of instances such as `Jobs` which is both `Person` and `Jobs(movie)`.

Table 3: Top-3 matched concepts based on correlation observed from seed matches. Prefix foaf: is http://xmlns.com/foaf/0.1/, d: is http://dbpedia.org/ontology/, y: is http://yago-knowledge.org/resource/.

DBpedia concepts	YAGO concepts	$corr()$
`foaf:Person`	`y:wordnet_person`	0.986
`d:Place`	`y:yagoGeoEntity`	0.951
`d:Organisation`	`y:wordnet_organization`	0.906
...	...	...

For each domain, we conducted feature-based blocking. Candidate degree threshold t was set to 10 in this experiment. Our blocking method showed near perfect reduction ratio (RR) in all domains (Table 4), which shows that the method has high effectiveness in reducing the search space for matching. Pair completeness (PC) is the upper bound to recall of alignment. PC was sufficiently high for the person and location domains, and ARIA achieved recall close to the bound obtained from PC. Note this bound is notably low for organizations due to feature sparsity, which explains low recalls of both ARIA and PARIS for this specific domain.

Table 4: Evaluation of feature-based blocking method for each sub-domain.

domain	#golds	RR	PC	f1.
person	846K	0.999	0.986	0.994
location	304K	0.998	0.928	0.962
organization	129K	0.999	0.832	0.909
etc	377K	0.999	0.826	0.905

Lastly, we evaluate the robustness of instance similarities between the result candidates of blocking methods for each domain (Table 5). We set triple similarity threshold θ as 0.8. ARIA was more accurate and faster than PARIS in every domain. ARIA increased overall precision by 18.6% and overall recall by 1.5% compared to PARIS. ARIA also significantly reduced response time by effectively reducing search space, whereas PARIS does not exploit blocking while adopting iterative approaches with high computational overheads.

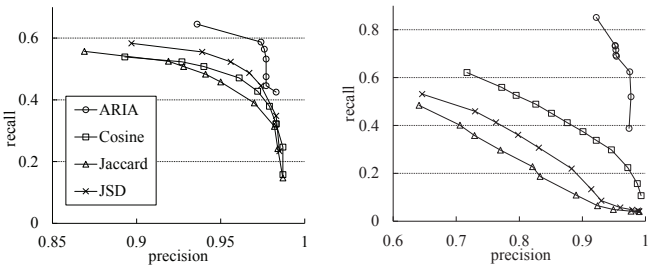
Similarity measure evaluation

We evaluate our proposed similarity measure and depict precision-recall graphs compared with three widely adopted measures– Jaccard coefficient, Cosine similarity with TF-IDF, and Jensen-Shannon divergence (JSD) (Figure 4). We use both DBpedia and DBpedia+ to observe the results over varying degrees of structure asymmetry.

Regardless of the degree of asymmetry, our proposed measure significantly outperforms all other measures. JSD showed accurate results in DBpedia, but this accuracy decreased significantly in the presence of asymmetry. Cosine similarity with TF-IDF showed reasonable performance in DBpedia+-YAGO task, because IDF weighing adjusts frequent features within DBpedia+. Our proposed measure dominated in both cases, because it exploits additional feature information in DBpedia+ to further improve accuracy.

Table 5: Instance alignment results using ARIA and PARIS on DBpedia-YAGO.

framework	domain	prec.	rec.	f1.	time
ARIA	per.	0.975	0.977	0.976	5 min
	loc.	0.963	0.905	0.933	3 min
	org.	0.966	0.815	0.884	2 min
	etc	0.981	0.702	0.819	6 min
	total	0.972	0.861	0.913	16 min
PARIS	per.	0.862	0.935	0.897	20 min
	loc.	0.791	0.832	0.811	22 min
	org.	0.898	0.809	0.851	2 min
	etc	0.666	0.762	0.710	50 min
	total	0.786	0.846	0.815	94 min



(a) DBpedia-YAGO (b) DBpedia+-YAGO

Figure 4: Precision-recall graphs of four matching similarity measures.

Conclusions

We studied the problem of aligning instances across KBs that are inherently asymmetric to each other. We identify three types of asymmetries, namely concept, feature, and structure asymmetries. We propose ARIA overcoming these three asymmetries, specifically by improving blocking methods to tolerate concept and feature asymmetries and extending similarity metric to aggregate only the symmetric structure. Compared to PARIS, a state-of-the-art method, our method improved precision by 19% and recall by 2%, and decreased processing time by more than 80%.

Acknowledgments

This research was supported and funded by Korea Institute of Science and Technology Information and Microsoft Research.

References

Baxter, R.; Christen, P.; and Churches, T. 2003. A comparison of fast blocking methods for record linkage. In *ACM SIGKDD*, volume 3, 25–27.

Biega, J.; Kuzey, E.; and Suchanek, F. M. 2013. Inside yogo2s: a transparent information extraction architecture. In

Proceedings of the 22nd international conference on World Wide Web companion, 325–328. International World Wide Web Conferences Steering Committee.

Bilenko, M.; Mooney, R.; Cohen, W.; Ravikumar, P.; and Fienberg, S. 2003. Adaptive name matching in information integration. *Intelligent Systems, IEEE* 18(5):16–23.

Euzenat, J., and Shvaiko, P. 2013. *Ontology matching*. Heidelberg (DE): Springer-Verlag, 2nd edition.

Gemmell, J.; Rubinstein, B. I.; and Chandra, A. K. 2011. Improving entity resolution with global constraints. *arXiv preprint arXiv:1108.6016*.

Hu, W.; Chen, J.; and Qu, Y. 2011. A self-training approach for resolving object coreference on the semantic web. In *Proceedings of the 20th international conference on World wide web*, 87–96. ACM.

Lacoste-Julien, S.; Palla, K.; Davies, A.; Kasneci, G.; Graepel, T.; and Ghahramani, Z. 2012. Sigma: Simple greedy matching for aligning large knowledge bases. *arXiv preprint arXiv:1207.4525*.

Lehmann, J.; Isele, R.; Jakob, M.; Jentzsch, A.; Kontokostas, D.; Mendes, P. N.; Hellmann, S.; Morsey, M.; van Kleef, P.; Auer, S.; and Bizer, C. 2014. DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web Journal*.

McNeill, N.; Kardes, H.; and Borthwick, A. 2012. Dynamic record blocking: efficient linking of massive databases in mapreduce.

Melnik, S.; Garcia-Molina, H.; and Rahm, E. 2002. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, 117–128. IEEE.

Qu, Y.; Hu, W.; and Cheng, G. 2006. Constructing virtual documents for ontology matching. In *Proceedings of the 15th international conference on World Wide Web*, 23–31. ACM.

Stoilos, G.; Stamou, G.; and Kollias, S. 2005. A string metric for ontology alignment. In *The Semantic Web–ISWC 2005*. Springer. 624–637.

Suchanek, F. M.; Abiteboul, S.; and Senellart, P. 2011. Paris: probabilistic alignment of relations, instances, and schema. *Proceedings of the VLDB Endowment* 5(3):157–168.

Winkler, W. E. 1999. The state of record linkage and current research problems. In *Statistical Research Division, US Census Bureau*.