Domain-Independent Entity Coreference in RDF Graphs

Lehigh University Technical Report LU-CSE-10-004
Dezhao Song and Jeff Heflin
{des308, heflin}@cse.lehigh.edu
Department of Computer Science and Engineering
19 Memorial Drive West, Bethlehem, PA 18015

Additional Key Words and Phrases: Entity Coreference, Semantic Web, Ontology, Domain-Independence, Discriminability

1. INTRODUCTION

The purpose of entity coreference[1] is to decide if two mentions of proper nouns refer to the same real world entity. A mention is an occurrence of a name in a document, a web page, etc. For example, in two documents (e.g., two news articles), two or more mentions of the name "James Henderson" may exist and an entity coreference algorithm can answer if they really identify the same real world person.

The entity coreference task is challenging primarily due to two general aspects: how to locate context information for each mention and how to utilize the context in an appropriate way. On one hand, we need to collect information for those different mentions. We can collect context information from the documents where the mentions occur. The Internet can be another source for finding context information.

On the other hand, it is really significant to use the context appropriately. There are various situations that can mislead the entity coreference results. Name variations, the use of abbreviations, and misspellings can all play a role in the final results. Also, the collected data may come from heterogeneous sources and may not be complete. For instance, in each of the two news articles, different aspects of James Henderson may present. One article may mention the name and affiliation while the other one can have some other information present, such as his name, date of birth, email address, etc. In addition to these, even though some information is present, it may be noisy data. For example, some date information is included in the context for Jame Henderson and it is treated as the date-of-birth of him; however, that date information could simply be the date of a social event that this Jame Henderson attended. An entity coreference algorithm needs to be able to deal with such problems and challenges.

Let's see some Semantic Web terminologies first. In Semantic Web, an ontology is an explicit and formal specification of a conceptualization, formally describing a domain of discourse. An ontology consists of a set of terms (classes) and the relationships (class hierarchies and predicates) between these terms. RDF is a graph-based data model for resources and relations between them. Two resources are connected via one or more predicates in the form of triple. A triple, ¡s, p, o¿,

[1]Entity coreference is also referred to as entity resolution, disambiguation, etc.

consists of three parts: subject, predicate and object. The subject is an identifier (e.g., an URI) and the object can either be an identifier or a literal value, such as strings, numbers and dates.

In this paper, we present a novel algorithm that focuses on entity coreference between ontology instances. Generally, given a pair of instances of comparable classes, our algorithm tells if this pair of instances are coreferent, referring to the same real world entity, such as the same person or publication. In our algorithm, for a given instance, we find its neighbourhood graph from the entire RDF graph through an expansion process. We learn the discriminability of each triple, taking into account its predicate and/or subject or object. Such discriminability is then discounted according to the triple's distance to the root node (the ontology instance). Through such a distance-based discounting approach and the learned discriminability, we learn the weight of a path from root to a RDF node in the neighbourhood graph (the context) of an ontology instance. Compared to systems that include a subset of these features, our proposed algorithm achieves the best performance on two types of ontology instances from two distinct datasets, with a F1-score ranging from 87% to 97%.

We organize the rest of the paper as following. Section 2 discusses related work to entity coreference. We then present two different discriminability learning schemes in section 3. Section 4 describes how we find the context information for an ontology instance. In section 5, we formally present our entity coreference algorithm and show the experiment results in section 6 together with some result analyses and discussions. Finally, in section 7 we talk about the conclusion and some possible future explorations.

2. RELATED WORK

Researchers have been working on entity coreference and similar topics for a long time. To solve the name disambiguation problem, researchers have developed a variety of string matching algorithms [Bilenko et al. 2003; Cohen et al. 2003], have tried to disambiguate similar names by exploiting the similarity of their contexts [Pedersen et al. 2005], and have explored applying relevant techniques to identify name equivalences in digital libraries [Feitelson 2004].

Some researchers have been working on entity coreference in free text. Bagga and Baldwin [Bagga and Baldwin 1998] employ a vector space model to do cross-document entity coreference on person mentions in free text. They first use an in-document coreference system to construct coreference chains within each document. A particular chain contains name mentions and pronouns that are coreferent. Then, for cross-document coreference, they utilize all the relevant sentences to a particular mention as context. The relevant sentences are those where a mention or its in-document coreferent mentions occur. The problem with this approach is that it relies on the in-document entity coreference system to give good results in order to collect descent context information. Gooi and Allan [Gooi and Allan 2004] employ three models, the incremental vector space model, the KL divergence and the agglomerative vector space model, for entity coreference on person mentions in free text. They use a window size of 55 words centered on a mention to collect its context information. They chose a 55-word window because their experiments showed

that the best results were achieved with such size. Mann and Yarowsky [Mann and Yarowsky 2003] utilize an unsupervised clustering technique over a feature space to do coreference. They extract information from web pages for each mention. The main difference is that they try to extract some more representative information from the web pages, such as biographical information, marriage and parent/child relationships and so on. Han et al. [Han et al. 2004] deploy two different models, the Naive Bayes classifier and the SVM, to disambiguate author names in citations. Given a citation, their algorithm predicts if it is authored by some certain author. They designed some features to fit into the classifiers; however, such features may not apply to other domains. Some graph based approaches have been employed as well to disambiguate mentions in social networks [Bekkerman and McCallum 2005] and emails [Minkov et al. 2006]. Other than pure free text, Wikipedia and Encyclopedic have also been used to find context information [Bunescu and Pasca 2006; Cucerzan 2007]. For example, special types of Wikipedia pages, such as the disambiguation pages and the embedded hyperlinks have been employed for exploiting context information. Named entity recognition [David and Satoshi 2007] can be treated as a preprocessing step for entity coreference. It recognizes different types of mentions, such as person, organization, etc. This technique is out of the scope of this paper.

Word sense disambiguation and duplicate record detection in databases are two closely related topics to entity coreference. A word can have multiple meanings while the task of word sense disambiguation is to choose the most appropriate one based upon the word's context [Yarowsky 1995], such as a piece of free text. For duplicate record detection in databases, it tries to detect duplicate tuples and remove redundancies [Elmagarmid et al. 2007]. Different database records can give the same information but are distinct in their representations. For example, different records can represent a person's name differently, in the formats of full name or first initial plus family name. Compared to word sense disambiguation, duplicate record detection can be more related and similar to the entity coreference problem.

As the emergence of the Semantic Web technologies, researchers have started showing interests in the entity coreference problem on the Semantic Web. Hassel, et al. [Hassell et al. 2006] propose an ontology-driven disambiguation algorithm. They try to match ontology instances from an ontology created from the DBLP bibliography [Ley 2002] to mentions in DBWorld documents[2]. They use the information provided in the triples of an ontology instance to match the context in free text. For example, if a person instance, named "John Smith", has affiliation information of "Stanford University" and in a DBWorld document, "John Smith" and "Stanford University" occur close to each other, then this adds some confidence that this person instance is coreferent to the name mention in the DBWorld document. Another clue is the co-occurrence relationship. For example, a person instance has the research interest "Artificial Intelligence" and if we can find the name of this person instance and the keywords "Artificial Intelligence" to co-occur (they do not need to be close to each other) in a DBWorld document, this also shows some possibility for the instance and the DBWorld mention to be coreferent.

[2]http://www.cs.wisc.edu/dbworld/

Their algorithm achieves good performance: 97.1% in precision and 79.1% in recall. The problem with this approach is that they manually and selectively picked some triples of the instances (e.g., name, affiliation, etc.) to use. The features (e.g., co-occurrence) were identified manually as well. However, for domains where domain expertise is difficult to obtain, it may not be feasible to choose what information would be important and useful; and it would also be difficult to identify useful features for those particular domains.

Different from previous papers, Aswani et al. [Aswani et al. 2006] propose an algorithm for matching two ontology instances. This can be particularly useful in connecting different Semantic Web information sources to make owl:sameAs statements, providing the opportunity to do data integration and to accomplish better query answering systems on the Semantic Web. Their algorithm matches person instances from an ontology converted from the BT digital library, containing 4429 publication instances and 9065 author names. One of their focuses is to exploit the web to find information to support the coreference process. For example, they issue queries with the family name of a person instance and the title of a publication instance to search engines and see if different author instances will finally come to have the same full name. A positive answer gives a hint to confirm that the two person instances are coreferent. Some other clues include finding the publication page for different person instances, measuring the similarities between the names of person instances and the titles of their publications. The authors test their algorithm on author instances with identical family names and achieve an accuracy from 90.48% to 100% given different datasets and features. Similar to the paper by Hassel et al. [Hassell et al. 2006], the feature set is manually identified. Also, some features require special management. For instance, the authors manually set up some rules to determine if a name is really a full name of a person or if a returned web page is really a person's publication page or simply a page from DBLP where papers of distinct authors may co-exist.

3. LEARNING DISCRIMINABILITY

In this section, we will present our approach for learning the discriminability of RDF triples, which partially answers the question that given some context information, how we could utilize it appropriately. We will address another aspect of this question in section 4.

Our discriminability learning approach is domain-independent. Given a specific dataset, without a domain-independent and automatic discriminability learning algorithm, we need to manually determine the importance of each triple. When disambiguating person instances, we need to manually determine that person name can be more discriminative than birthplace or hometown, and others like such. Our approach, given a new dataset, takes the entire dataset (triples) as input and automatically learn the discriminabilities regardless of the domain of that dataset, such as academia domain or some others.

Generally, each triple has its own importance, reflecting its possible level of discrimination, to the ontology instance from which it originally comes from. This is related to finding the neighborhood graph for an instance which we will formally introduce in section 4.

3.1 Predicate Discriminability

We call the first type of discriminability, the Predicate Discriminability. Thinking broadly, we can measure the discriminability of a triple by only looking at what its predicate is. As for a predicate, the more diverse value set it has, the more discriminative it will be. Triples with different predicates, such as "has-publication-date" and "has-author" could have different discriminabilities. Equations 1 and 2 show how we compute predicate discriminability.

$$Per_{p_i} = \frac{SizeOf(distinct\ object\ of\ p_i)}{number\ of\ occurrences\ of\ p_i} \tag{1}$$

where Per_{p_i} represents a percentage value for predicate p_i, which is the size of p_i's distinct object value set divided by its number of occurrences in the entire dataset. We record the max percentage value as Per_{max}. We then normalize such values so that the most discriminative predicate has a discriminability of 1. The normalization is shown in equation 2

$$P_{p_i} = \frac{Per_{p_i}}{Per_{max}} \tag{2}$$

where P_{p_i} is the predicate discriminability for predicate p_i.

Depending on what category of an object is being compared, a predicate may be used in the subject-to-object direction or in the object-to-subject direction. A predicate that discriminates well in one direction may not do well in the other. This is related to how we expand an instance to collect neighbourhood triples in section 4. Basically, when we do the expansion, we use an URI both as the subject and the object, so a predicate has different discriminabilities to two directions.

To clearly represent discriminabilities, for a given predicate p_i, we use Per_{p_i} and $Per^-_{p_i}$ to denote the percentage values to the object and subject direction respectively; then the predicate discriminabilities to the two directions are denoted as P_{p_i} and $P^-_{p_i}$. Equations 1 and 2 compute predicate discriminabilities to the object direction. The discriminabilities to the subject direction can be computed in the same way by replacing appropriate variables.

Take the "has-author" predicate (with domain of publication class and range of person class) as an example. It has P_{p_i} of 0.63 and $P^-_{p_i}$ of 0.398. So, when disambiguating between publication instances, having a common author can be more discriminative than having a common publication when doing coreference on person instances.

The intuition behind our predicate discriminability is that the discriminability of a triple is determined by its predicate. And such discriminability will then contribute to the entity resolution process. For example, if two publication instances happen to have the triples with the same object value via predicate "has-publication-year" that only has a weight of 0.05, then such a coincidence does not really add much value to determine they are coreferent; however, predicate "has-author" shows a much higher discriminability to the object direction (0.63), so that having equivalent object values for this predicate will give us a better idea that these two publications be coreferent.

Currently, when counting the size of distinct object/subject value set, we assume that if any two objects/subjects are distinct, then they truly represent different

things. However, they could actually represent the same real world entity. With such unknown coreferent objects/subjects, we are actually overestimating the discriminability. But if we assume that for every predicate such unknown coreferent relationships occur uniformly throughout the dataset, we actually overestimate all predicates by the same proportion. Thus our current approach still gives reasonable discriminability.

3.2 Value-Dependent Discriminability

The first scheme captures the discriminability of every predicate in the context of the entire RDF graph of a dataset. In this circumstance, triples with different objects or subjects but having the same predicate, will have the same discriminability, which seems to lose the speciality of those different objects or subjects. In attempting to bring in such specialities, we propose this value-dependent discriminability which captures the discriminability of a triple by considering its (predicate, object/subject) pair.

Let's perceive a direct and clear view of the differences between the two schemes via a concrete example. Given a predicate, "has-fullname", two triples present in the RKB dataset with two different objects (definitely there are more): "James Smith" and "Malte Appeltauer". To the object direction (these names are literals, so that they can only occur as objects), these two triples have the same predicate discriminability, which is 0.35; however, for value-dependent discriminability, the triple with "James Smith" has a discriminability of 0.0184 while the other triple's discriminability is 0.175. Such a difference shows that the name of "James Smith" is more common within this dataset.

Equations 3, 4 and 5 show how we compute this value-dependent discriminability to the object direction.

$$OCC_{(p_i,x)} = |\{t \epsilon G | t =< any\_sub,\ p_i,\ x >\}| \tag{3}$$

where t is a triple, any_sub can be any subject value, p_i is the predicate, x represents a particular object node, and G is the entire RDF graph. It gives the number of occurrences that a particular (predicate, object) pair occurs in the entire RDF graph. Equation 4 shows how we normalize such occurrence value to get a ratio.

$$R_{(p_i,x)} = \frac{MINOCCp_i}{OCC_{(p_i,x)}} \tag{4}$$

where $MINOCCp_i$ represents the minimum number of occurrence of an object for predicate p_i and $R_{(p_i,x)}$ is the normalized ratio of the pair of (p_i, x). Finally, equation 5 shows how we calculate our value-dependent discriminability.

$$V_{(p_i,x)} = P_{p_i} * R_{(p_i,x)} \tag{5}$$

where P_{p_i} is the predicate discriminability of predicate p and $R_{(p_i,x)}$ is the ratio from equation 4.

Intuitively, value-dependent discriminability says that the more often a particular pair of (predicate, object) occurs, the less discriminative a triple with this predicate and this object will be. We cannot simply use the ratio as the discriminability since it does not capture the context of the entire RDF graph. For example, for predicate p, its object values can be uniformly distributed but occur very frequently as well.

So, each object value of p will have ratio of 1 but the predicate discriminability is low. With equation 5, we try to use the discriminability of this predicate to help giving a better value-dependent discriminability of a (predicate, object) pair.

Again, to different directions, a triple can have different discriminabilities. Equations 6 shows how to calculate the OCC value to the subject direction and the discriminability can be computed similarly by replacing appropriate variables in equations 4 and 5.

$$OCC^{-}_{(p_i,x)} = |\{t \epsilon G | t =< x,\ p_i,\ any\_obj >\}| \tag{6}$$

The difference is that we change the triple format to $t < x,\ p_i,\ any\_obj >$, meaning that x (the subject) is what we care about now. Similarly, we use $V^{-}_{(p_i,x)}$ to denote the value-dependent discriminability of a (predicate, subject) pair.

4. FIND NEIGHBORHOOD GRAPH

In this section, we address another challenge in developing an entity coreference algorithm, which is how to locate the context for the mentions and we also give a complete answer to the question that how to utilize the context appropriately.

In this paper, we only use the RDF graph to locate context information. We collect paths in the RDF graph within a certain distance to an ontology instance (the root node) that we do coreference on. Starting from the root node, we search triples whose subject or object equals to the URI of this instance and record those expanded triples in database. With those expanded triples, if the objects or subjects are still URIs or blank nodes, then we repeat this search or expansion process on them to get further expanded triples until we reach the depth limit or a literal value. With our expansion process, we end up having a set of paths for each ontology instance, starting from that instance and ending with an URI or a literal value. When we end on a blank node, we do not record that path because we cannot simply compare two blank nodes and see if they are identical. But we rely on paths that go through blank nodes to get further literals and URIs. We define a path as all the nodes and predicates in an expansion chain as shown in Equation 7.

$$path = (n_0, p_1, n_1, ..., p_n, n_n) \tag{7}$$

where n_0 is the root node, $n_i (i > 0)$ is any other expanded RDF node and p_i is a predicate in the path. We treat the set of paths as the context of an ontology instance, denoted as $N(i)$, where i represents the instance. We denote the RDF node at the rear of a path as $End(p)$ where p is a path and $p \epsilon N(i)$.

Together with the learned discriminability introduced in section 3, we assign each path in the neighbourhood graph a weight, indicating its importance to the root node. The weights combine two elements, the learned discriminability and a discount value. Figure 1 displays how we assign the weight to each path.

In figure 1, starting from root, we get to node 1, 2 and 3 and we reach node 4, 5, 6, 7 by further expanding node 2, so on and so forth. PV_1 and PV_2 represent the discriminabilities for the two triples ending with node 2 and 5 respectively. Here, we use PV together to represent either predicate or value-dependent discriminability generically. We also add another parameter to each node called factor, indicating how important a node is to its father node. For example, F_1 is the factor of node 2

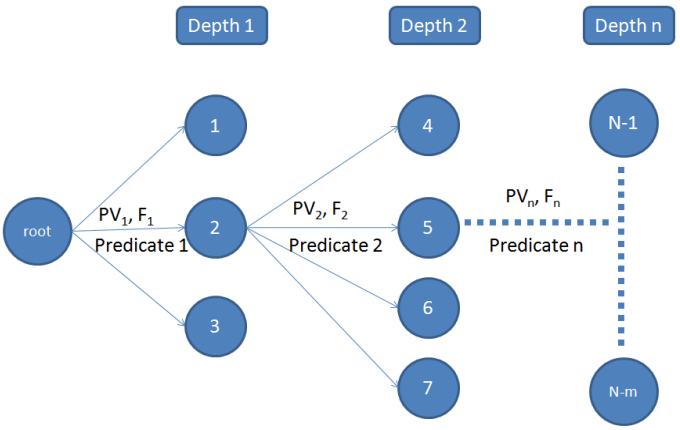

Fig. 1. Assign Weight to Neighbourhood Graph

to the root node and its value is one-third because three triples get expanded from the root. Each of the three nodes (node 1, 2 and 3) only represents one-third of their father node conceptually. The underlying semantics of this factor is to portion out the importance of one node to its expanded nodes in an equal manner.

With the factors and the learned discriminability, we take a distance-based discounting approach to assign a weight to a path in the neighbourhood graph. Equation 8 shows how to calculate this weight.

$$W_{path} = \prod_{i=1}^{length(path)} PV_i * F_i \tag{8}$$

where the length function counts the length a path, PV_i and F_i represent the discriminability and factor for each triple in the path respectively. The intuition here is that as we expand further in the RDF graph, more noisy data could be introduced. In order not to overwhelm the context, the discriminability of each expanded triple should be appropriately adjusted. We call this a distance-based discounting approach.

In order for the expansion to end, we need to set a depth limit. In this paper, we set this value to 2. With this limit, we've discovered that it is sufficient to get enough context information from the entire RDF graph. For example, in the RKB dataset[3], given a person instance, we can find its name, affiliation information, and the URIs of this person's publication instances at depth 1; going further to depth 2, we will have the titles and the dates of those publications and the URIs of the coauthors on those publications, etc.

5. ENTITY COREFERENCE ALGORITHM

We find the context for an instance and adopt a distance-based discounting methodology to appropriately assign weights to paths in the context. In this section, we

[3]This is one of the datasets that we use for evaluation.

formally present our entity coreference algorithm for ontology instances with the weights.

5.1 Algorithm Design

Algorithm 1 presents the pseudo code of our entity coreference algorithm for ontology instances. In this description, a and b are two ontology instances; $N(G, a)$ returns the context (a set of weighted paths) of instance a in RDF graph G; $E(path)$ returns the end node of a path; the function $PathComparable$ tells if two paths are comparable; Sim is a string matching algorithm that computes the similarity score between two literals.

Algorithm 1 Compare(N_a, N_b), N_a is the context N(G,a) and N_b is N(G,b); returns a float number (the similarity of a and b)

1. $total\_score \leftarrow 0, total\_weight \leftarrow 0$
2. **for all** *paths* $m \in N_a$ **do**
3. **if** $\exists path\ n \in N_b$, $PathComparable(m, n)$ **then**
4. $path\_score \leftarrow 0, path\_weight \leftarrow 0$
5. **if** $E(m)$ *is literal* **then**
6. $path\_score \leftarrow \max_{n' \in N_b, PathComparable(m,n')} Sim(E(m), E(n'))$
7. /* path n' has the highest score with m */
8. $path\_weight \leftarrow (W_m + W_{n'})/2$
9. **else if** $E(m)$ *is* URI **then**
10. **if** $\exists path\ n' \in N_b, PathComparable(m, n'), E(m) = E(n')$ **then**
11. $path\_score \leftarrow 1$
12. /* path n' has identical end node with m */
13. $path\_weight \leftarrow (W_m + W_{n'})/2$
14. **end if**
15. **end if**
16. $total\_score \leftarrow total\_score + path\_score * path\_weight$
17. $total\_weight \leftarrow total\_weight + path\_weight$
18. **end if**
19. **end for**
20. **return** $total\_score/total\_weight$

The essential idea of our entity coreference algorithm is that we adopt the bag-of-paths approach to compare paths between ontology instances, such as instanceA and instanceB. For each path (pathA) of instanceA, we compare its rear node to the rear node of every path of instanceB with a comparable sequence of predicates and choose the highest similarity score, denoted as path_scoreA. Also, we need to determine the weight of this path_score. Here, when considering the weight, we need to take into account the weight (weightA) of pathA and the weight (weightB) of the path of instanceB with which pathA has the highest similarity score. Then we use the average of weightA and weightB as the path_weight for path_scoreA. We need to repeat the process for every path of instanceA. With the pairs of (path_score, path_weight) for a pair of instances, we calculate their weighted average in order to have the final similarity measure between instanceA and instanceB. The same

process is repeated for all pairs of ontology instances of comparable categories, i.e., person-to-person and publication-to-publication. In the last line of the pseudo code, "sim" represents the similarity scores for all pairs of instances in the list.

5.2 Predicate Comparability

As we described, we will compare instanceA's paths with paths of instanceB, which brings in the question that if the sequences of predicates of two paths from the two instances are comparable. For example, predicates "has-publication-date" and "has-birth-date" are not comparable, though both of their object values represent dates information. In some situations, the comparability of predicates is not very clear. For instance, the following two predicates "author-on" and "edit-on" can be vague in their comparability. For a publication, a person that did some edits on it does not necessarily have to be listed as an author of it. It depends on what the actual situation is. In such a circumstance, without letting such two predicates comparable, we could lose some portion of the final results; on the other hand, adding them in might hurt the precision. Determining the comparability of predicates is not always an easy job, particularly for a sequence of predicates.

In our algorithm, we manually identified the comparability of predicates. We only make two or more predicates comparable if we can verify that they truly have the same underlying semantics in the ontologies that our datasets commit to. We look at information from the ontologies and examine the data to see if two predicates can be comparable. For example, the following two predicates are comparable "full-name" and "pretty-name".

Furthermore, in our current implementation, we only check the comparability of the last predicate in the path while ignoring others. We treat the last predicate in a path as a composition predicate for all the predicates in that path. We admit that this is not the ideal solution and in future work we will explore into the comparability of a sequence of predicates by examining each predicate in the path. Also, manually determining comparability is not good enough. Utilizing the information from the ontologies and the dataset, the comparability of predicates could be exploited. For example, two equivalent predicates in the ontology should be comparable. Ontology alignment, a well studied topic in Semantic Web, can be helpful in automatically determining the comparability of predicates between multiple ontologies. It is out of the scope of this paper.

5.3 Compare URIs and Literals

Given the condition that two paths are comparable, if the two rear nodes are two literal values, e.g., two person names or two publication titles, then we adopt the JaroWinklerTFIDF string matching algorithm [Cohen et al. 2003].

In another situation, if the two nodes are both URIs in the RDF graph, then we will simply compare them to see if they are identical. The similarity score between two literals ranges from 0 to 1 while the score between any pair of URIs will be either 0 (not match) or 1 (identical). If the URIs do not match, the similarity is computed by further expanding those URIs as we presented in section 4. When determining the comparability of the paths, we will not allow two paths whose last predicates that have opposite underlying semantics to be comparable, so that we won't have the situations where we compare an URI object to a literal object.

5.4 The Open World Problem

Another challenge that we face is that we cannot make a closed-world assumption so that we need to deal with open-world. We cannot assume something we don't know to be false, everything we don't know is undefined.

On one hand, within the entire dataset, some information can be missing. Our RKB dataset is composed of several subsets of the complete RKB dataset, such as ACM, IEEE, DBLP, CiteSeer, etc. Each of them, in the case of person instance, only includes a certain portion of his/her information, such as they can only contain some of this person's publications.

In our entity coreference algorithm, we try to relieve this Open World problem. First of all, we do not apply penalties to mismatches on URIs. This means that if the rear node of pathA of instanceA is an URI but it doesn't match any rear node of paths with comparable predicates of instanceB, we do not add any weight to the total_weight. These mismatched URIs are expanded to get further literals and other URIs, which will be used to determine the similarity.

Second, we do not apply any penalties on missing information. If there isn't any path of instanceB that has comparable predicate for pathA of instanceA, still we do not apply any penalties. The intuition behind our approach is that we compare every path present in the context and apply appropriate penalties; in the meanwhile, any mismatches that are caused by information incompleteness cannot simply be treated as real mismatches. Our current approach may not be best solution to the open world problem and we expect to better handle it in the future.

6. EVALUATION

In this section, we present how we collect and prepare our datasets, the evaluation metrics and methodology we use and the experiment results by applying our entity coreference algorithm on two instance categories in two distinct datasets.

6.1 Experiment Setup

6.1.1 *Data Collection and Preparation.* We evaluate our entity coreference algorithm on instances from two distinct datasets, the RKB dataset and the SWAT dataset.

The RKB Dataset. This dataset is available from the RKB Explorer homepage[4], which is part of ReSIST "Network of Excellence" [Glaser et al. 2008]. The entire RKB dataset consists of 54 subsets, containing academic publications from different sources, such as ACM, CiteSeer, DBLP, and so on. Since the entire RKB dataset is quite large, we pick eight subsets (in RDF format) of it: ACM, DBLP, CiteSeer, EPrints, IEEE, LAAS-CNRS, Newcastle (University of Newcastle upon Tyne), ECS. For convenience, in the rest of this paper, we call these eight subsets together as the RKB dataset. This dataset has 82 million triples (duplicates are removed), 3,986,676 person instances and 2,664,788 publication instances.

The SWAT Dataset. Another dataset is the SWAT dataset. It consists of the data parsed from the downloaded XML files of CiteSeer and DBLP. We implement a program that transforms the original XML files into RDF format. The converted

[4]http://www.rkbexplorer.com/data/

RDF files are then parsed and stored into database, resulting in a total of 26 million triples. Within our SWAT dataset, there are 904,211 person instances and 1,532,758 publication instances.

Although the two datasets share some information, the main difference is that they use different ontologies, so that different predicates are involved. Their coverage of publications could also be different. Additionally, some information may be ignored from the original XML files for SWAT dataset during transformation. One thing to note is that when we parse the RDF data into database, we ignore all the owl:sameAs statements for both datasets. So, as seen in section 4, the owl:sameAs statements will not help in connecting two coreferent instances (with distinct URIs) through the expansion process. They are only used for evaluating our results but not for facilitating our entity coreference process in any sense.

Test Set Preparation. In each dataset, there are different classes of ontology instances, such as person, publication, organization, etc. We evaluate our entity coreference algorithm on person and publication instances from the RKB dataset and person instances from the SWAT dataset. Also, because there exist millions of instances in one dataset of one single instance category, we only select a small but reasonable amount of them for our evaluation.

In order to increase the ambiguity of our test sets, we randomly picked 1,601 person instances from the RKB dataset through a filtering process. If the names of two person instances have a similarity score higher than 0.5 but they are still said not to be coreferent based upon the groundtruth, we will put this pair of instances into an instance pool. Similarly, with the comparison of publication titles, we collected 2,102 publication instances from the RKB dataset. For the SWAT dataset, similarly, we get an instance pool of 1010 person instances. We did not do experiments on SWAT publication instances because we did not have enough time to generate the groundtruth for publication instances of the SWAT dataset. We will describe how we achieve the goldstandard in section 6.1.2. There are 393339, 78444 and 52003 triples for our RKB publication, RKB person and SWAT person test sets respectively. Note that, in the experiments we are not only comparing those instances whose names or titles have similarity scores higher than 0.5 but not coreferent. We apply our algorithm on every pair of instances in each of the three test sets.

6.1.2 *Evaluation Metric and Methodology.* In our evaluations, we use the standard measures, precision, recall and F1-score, from Information Retrieval. As we presented, our algorithm does entity coreference on every pair of instances and stores results into database in the form of (instanceA, instanceB, score). Precision is measured as the number of correctly detected coreferent pairs divided by the total number of detected pairs given some threshold t as shown in equation 9. Recall is defined as the number of correctly detected coreferent pairs divided by the total number of coreferent pairs given a test set of ontology instances as shown in equation 10. To give a comprehensive view of our results, we choose F1-score which is defined in equation 11

$$Precision_t = \frac{|correctly\ detected|}{|totally\ detected|} \tag{9}$$

$$Recall_t = \frac{|correctly\ detected|}{|total\ number\ of\ coreferent\ pairs\ for\ test\ set|} \tag{10}$$

$$F1\text{-}Score_t = 2 * \frac{Precision_t * Recall_t}{Precision_t + Recall_t} \tag{11}$$

where t represents threshold in all the above three equations.

There are a few things to note about measuring precision and recall for our entity coreference algorithm. First of all, collecting groundtruth is a very challenging task, requiring manual effort to ensure quality. In this paper, we evaluate our algorithm on different instance categories from two different datasets. In the RKB dataset, the ReSIST project provides an online service, called crs, which provides the coreference groundtruth. In order to retrieve the goldstandard, people can issue queries that include the URI of an ontology instance to the online service and it will return the coreference information for this particular instance in RDF format. With such a crs service, we use the URI of every ontology instance in our test sets (person instances and publication instances) as the parameter and crawl those RDF files containing coreference groundtruth. Then we parse these results into database in form of (instanceA, instanceB), meaning that these two instances are coreferent. To verify soundness of the groundtruth crawled from RKB, we manually verified 300 coreferent pairs of person instances and publication instances respectively for our entire RKB dataset, while there are 81,556 and 148,409 in total for person and publication instances each (before reasoning). We admit that we only verified a small portion of the groundtruth for soundness but none of them appears to be wrong. For the SWAT dataset, the authors hand-labeled the groundtruth for the person instance test set.

Furthermore, different from entity coreference algorithms with free text, we need to take care of the transitivity between ontology instances. This says that if instanceA is coreferent with instanceB which is also coreferent with instanceC, we come to the conclusion that instanceA and instanceC are coreferent as well due to transitivity. In order to give the best correct evaluation results, we implemented a reasoning algorithm that materializes all the coreferent pairs that can be achieved through reasoning on transitivity. Note that, we do not materialize reflexivity and symmetry.

Finally, in order to guarantee completeness for the RKB groundtruth, we adopted a lazy or passive approach to verify the completeness for both person and publication instances groundtruth. We run our entity coreference algorithm on these two test sets and achieve the results. We then apply 7 different thresholds, ranging from 0.3 to 0.9, to evaluate the results based upon the groundtruth that we have achieved through the crawling and reasoning steps. After this, we pick the comparison system (we will formally present our comparison systems in section 6.1.3) that obtains the lowest precision, find out all the pairs that are detected by this system but are said not to be coreferent according to the groundtruth. For these "wrongly" detected pairs, we manually check each of them to see if any pair should be coreferent.

To manually check those pairs, we first search for all the triples related to the two instances from the entire RDF graph and check them on the Internet if necessarily using authors' DBLP homepages and their real homepages. Through this lazy-

Table I. Comparision Systems

System	Expansion Depth	P	V	D
E1	1			
E1-P	1	*		
E2	2			
E2-P	2	*		
E2-D	2			*
E2-P-D	2	*		*
E1-V	1		*	
E2-V	2		*	
E2-V-D	2		*	*

verification step, we were able to find out 295 missing coreferent pairs for the RKB person test set. Mostly, the reason for RKB crs service to miss a coreferent pair of person instances is that instances from different subsets of RKB share little common publications. Note that our 1,601 RKB person test set is generated after this lazy-verification step. Before, there were 1,787 person instances.

6.1.3 *Comparison Systems.* In order to show the effectiveness of our proposed entity coreference algorithm, we compare our proposed algorithm to some comparison systems that are not equipped with all the features we have presented. In our proposed system, we have the following features: expansion (E#) (# represents the depth of expansion), discriminability (P or V, representing predicate discriminability or value-dependent discriminability), discount (D) which is implemented by using the factor. The comparison systems are shown in table I.

For example, the comparison system E2-D says it expands to depth 2, doesn't use the learned discriminability but uses the discounts of each expanded triple. So, for E2-D, equation 8 will change to equation 12.

$$W_{path} = \prod_{i=1}^{length(path)} F_i \tag{12}$$

where path length is 2.

For system E2-P, it uses predicate discriminability in the way that it propagates the discriminabilities of the triples along the expansion chain. Although such propagations can be viewed as one type of discount, our real discouting is from the factors. So, for this sytem, the weight of a path is computed with equation 13.

$$W_{path} = \prod_{i=1}^{length(path)} P_i \tag{13}$$

where P_i is the predicate discriminability of a triple at depth i. Future work can be exploring not propagating the discriminabilities and using the first predicate in an expansion chain as the composition predicate. For systems E1 and E2, the weight for every path in the neighbourhood graph is set to 1. Our proposed algorithm then uses depth 2 expansion, includes discounts and either predicate or value-dependent discriminability to form path weight.

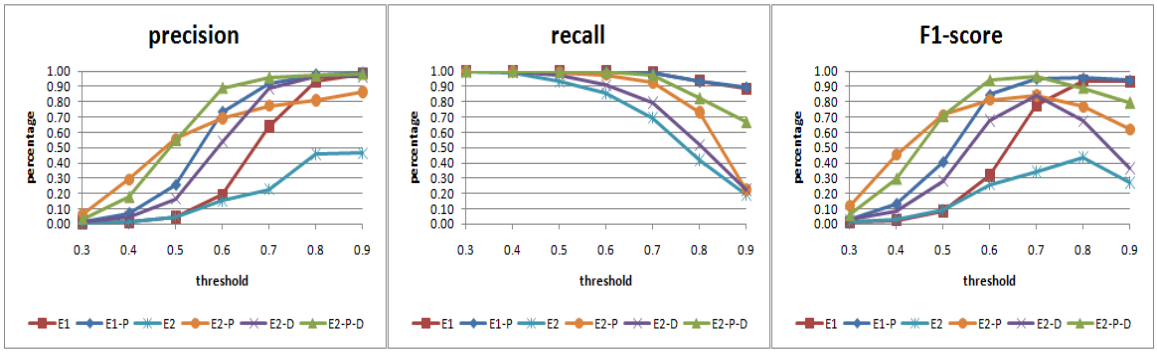

Fig. 2. Entity Coreference Results for RKB Publication Instances

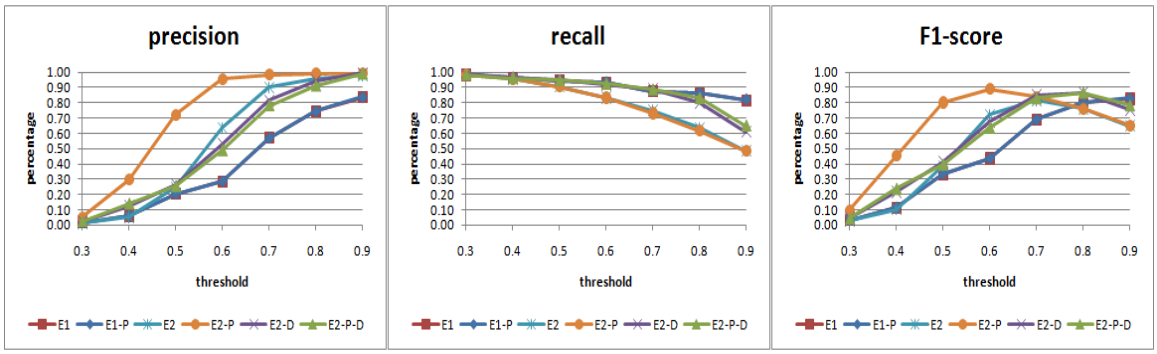

Fig. 3. Entity Coreference Results for RKB Person Instances

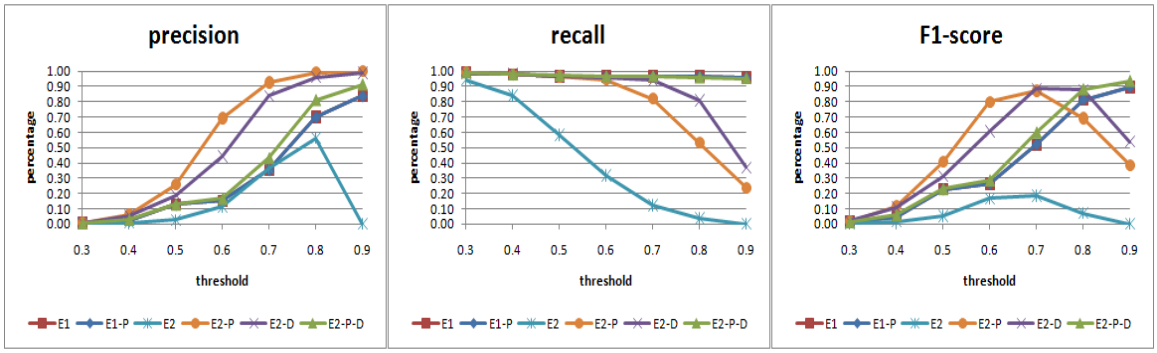

Fig. 4. Entity Coreference Results for SWAT Person Instances

6.2 Experiment Results And Discussions

Figure 2, 3 and 4 show the precision, recall and F1-score for results of RKB publication instances, RKB person instances and SWAT person instances respectively.

In these three experiments, we adopt the predicate discriminability. From the three F1-scores, we can see that our distance-based discounting entity coreference algorithm (E2-P-D) achieves the best or nearly the best performance. Compared to E2-P, it is 2% lower than E2-P's best F1-score, at threshold 0.6 for E2-P and 0.8 for E2-P-D respectively, on the RKB person instances but are clearly better in the other two test sets at high thresholds. The F1-score that our algorithm achieves for RKB publication, RKB person and SWAT person is 96.7%, 86.7% and 93.5% at threshold 0.7, 0.8 and 0.9 respectively.

System E1 is our baseline system in the sense that there is no discriminability included, no discounts at all and that it only considers adjacent triples. Compared to E1, E2 finds neighbourhood graph at one more depth so it has more context. But without discounts and discriminability, it is clearly worse than E1 for RKB publication and SWAT people; however, interestingly for RKB person, E2 performs better than E1 from threshold 0.3 to 0.7 but is surpassed by E1 at thresholds 0.8 and 0.9. We will explore into this issue in the future.

Comparing E1 and E1-P gives us an idea of how effective the predicate discriminability can be by itself. Note that sometimes the curve for E1-P or E1 is not clear because they are overlapping. The results show that in all cases, E1-P is better than or as good as E1. Furthermore, E2-P is better than E2 for RKB publication and SWAT people at all thresholds, showing the effectiveness of using predicate discriminability in a broader context, which in turn shows the contribution of doing expansion. For RKB person, E2-P is better than E2 at low thresholds from 0.3 to 0.6 but they start to have relatively the same performance from thresholds 0.7 to 0.9.

The difference between E2 and E2-D states that by applying factor discounts only can also gives us a significant improvement particularly for RKB publication instances and SWAT person instances. There is still some improvement for RKB person instances but not as significant as shown in the other two test sets.

Last, our proposed algorithm, E2-P-D, is able to achieve the best performance for RKB publications and SWAT person instances but not for RKB person instances. Our algorithm has low precision at low thresholds though it goes up as threshold goes higher. This can be partially because we don't apply penalties on URI mismatches and missing literal information. Most of the comparison systems experience significant drops in recall at higher thresholds, but E2-P-D is less affected by applying high thresholds. Compared to E2-D, E2-P-D shows better results for RKB publications at all thresholds and SWAT person at threshold 0.9. This verifies the effectiveness of using predicate discriminability. When comparing to E2-P, E2-P-D shows significant improvement for RKB publications from thresholds 0.6 to 0.9 and SWAT person instances at thresholds 0.8 and 0.9. Additionally, for both RKB publications and SWAT person instances, E2-P-D is not dominating at low thresholds but it is able to be on top at high thresholds.

We also did experiments with our value-dependent discriminability, the last three systems in table I; however, with that, our proposed algorithm (E2-V-D) achieves the best performance (F1-score) only for the SWAT person test set which is still lower than with predicate discriminability. For RKB publication, RKB person, and SWAT person instances respectively, their best F1-scores drop to 91.8% at

threshold 0.7, 81.2% at threshold 0.7 and 90.9% at threshold 0.9.

In order not to overwhelm the readers, we do not exhibit the figures for coreference results with the value-dependent discriminability but we will give some comparisons between the two types of discriminabilities. Comparing performance between systems that use predicate discriminability and value-dependent discriminability, the best F1-scores of E1-V are 95.1% at threshold 0.8 for RKB publication, 83% at threshold 0.9 for RKB person, 90.9% for SWAT person instances at threshold 0.9, which are nearly as good as E1-P. The best F1-scores of E2-V are 91.8% for RKB publications compared to 84.2% of E2-P, 85.0% for RKB person instances compared to 88.9% of E2-P and 81.2% for SWAT person instances compared to 87.1% of E2-P; E2-V-D has its best F1-scores of 91.8% for RKB publications, 81.2% for RKB person instances and 90.9% for SWAT person which are lower than with predicate discriminability again. Based upon such comparisons, we can see that, mostly, value-discriminability does not give better results and when we combine discriminability with discount together, it is significantly worse than predicate discriminability.

The results show certain advantages of our approach; however, there are a few points to discuss. First of all, with value-dependent discriminability, our proposed algorithm is not able to achieve the best performance. The idea to have such a discriminability is to capture the discriminability of different object values. We were expecting to achieve some better results. The reason for such unexpected results might be that the distribution of object values within a predicate adds some negative affects to the discriminability value. As part of future work, we expect to have a better approach to capture the differences between objects or subjects.

It is also more time-consuming to compute the value-dependent discriminability. For the RKB dataset, it will take about 2.5 hours finishing computing our value-dependent discriminability to the object direction; while computing predicate discriminability only takes 15 minutes. There are 53 predicates in our RKB dataset. For value-dependent discriminability, we need to take care of each (predicate, object) pair and store the results into database, so that both the computing and storing processes take time. There are 15 millions records in the value-dependent discriminability table; while there are only 53 records in the predicate discriminability table to the object direction.

Another issue that comes out is that as we apply higher threshold, recall generally goes down for all systems. This leaves us the question that can we improve this? As we described in section 5.4, we are facing the Open World problem. Different subsets of RKB or SWAT may not have complete information for an ontology instance. So, two person instances from ACM RKB and DBLP RKB can be filtered out by applying a high threshold because both of their contexts miss certain amount of information. One possible solution to this problem is to merge the contexts of two instances when we have a very high similarity measure for them during the coreference process. The intuition behind such merging is to let the context to evolve to be more complete. So, when we continue to compare the merged context with others', we then reduce the chance to have mismatches caused by information incompleteness in heterogeneous information sources. But we need to be very careful about doing such merges, since it is easy to bring in noises if the

standard for doing merges is not strict enough.

Currently, we do not apply penalties for URI mismatches or missing information. However, applying penalties in such scenarios may cause us to have lower recall. So it is always the problem of keeping a balance between precision and recall. Our choice is not to sacrifice recall while still having a good control on precision by exploiting appropriate weights and context information. One possible way to apply penalties on those situations might be to employ some iterative entity coreference algorithm. At each pass, we record the instance pairs that are clearly not coreferent or clearly coreferent (depending on how the algorithm is designed) and integrate the intermediate results into further iterations until we are only gaining new results under some pre-defined level.

A final note is that our bag-of-paths approach can be misleading the results in some particular cases in the sense that it ignores some underlying semantics. Take the predicate "has-date" as an example. The object of this predicate will only be some literal values, representing dates. However, this doesn't necessarily mean that we can always compare the two rear nodes of two paths that both have this predicate as their last predicate. We also need to take care of the semantics from the subject point of view. For all the expanded triples, to the object direction, we ignore their real subjects but only using the predicates and objects to form the path. So, we might have two triples in the context both with the "has-date" predicate but one has its original subject as a social event while the other has its original subject as a person. In this case, we should not compare the two date literal values because, essentially, they are telling different semantics.

Although the effectiveness of our proposed algorithm has been clearly and confidently verified, we realize that there are still some interesting open questions or challenges left. We will continue to explore appropriate approaches to handle those problems.

7. CONCLUSION AND FUTURE WORK

In this paper, we propose an entity coreference algorithm for Semantic Web instances. Our algorithm finds a neighbourhood graph of an ontology instance as its context information. With two different types of discriminability learning schemes and our distance-based discounting approach, we assign a weight to each path in the context. We adopt a bag-of-paths approach to compute the similarity measure between ontology instances pairwisely. Our entity corefernce algorithm is verified with three test sets and it achieves the best performance on two of them.

Some interesting challenges are still left with our current algorithm. For future work, we plan to try some iterative entity coreference algorithm in order to apply appropriate penalties on URI mismatches. Also, without taking the current bag-of-paths approach, we are interested in exploring some graph-based matching algorithm because, essentially, the context for an ontology instance is not a set of paths but a graph.

REFERENCES

Aswani, N., Bontcheva, K., and Cunningham, H. 2006. Mining information for instance unification. In *International Semantic Web Conference*. 329–342.

Bagga, A. and Baldwin, B. 1998. Entity-based cross-document coreferencing using the vector space model. In *COLING-ACL*. 79–85.

Bekkerman, R. and McCallum, A. 2005. Disambiguating web appearances of people in a social network. In *WWW*. 463–470.

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

Bunescu, R. C. and Pasca, M. 2006. Using encyclopedic knowledge for named entity disambiguation. In *EACL*.

Cohen, W. W., Ravikumar, P. D., and Fienberg, S. E. 2003. A comparison of string distance metrics for name-matching tasks. In *IIWeb*. 73–78.

Cucerzan, S. 2007. Large-scale named entity disambiguation based on Wikipedia data. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. Association for Computational Linguistics, Prague, Czech Republic, 708–716.

David, N. and Satoshi, S. 2007. A survey of named entity recognition and classification. *Linguisticae Investigationes 30,* 1 (January), 3–26.

Elmagarmid, A. K., Ipeirotis, P. G., and Verykios, V. S. 2007. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng. 19,* 1, 1–16.

Feitelson, D. G. 2004. On identifying name equivalences in digital libraries. *Inf. Res. 9,* 4.

Glaser, H., Millard, I., and Jaffri, A. 2008. Rkbexplorer.com: A knowledge driven infrastructure for linked data providers. In *ESWC*. 797–801.

Gooi, C. H. and Allan, J. 2004. Cross-document coreference on a large scale corpus. In *HLT-NAACL*. 9–16.

Han, H., Giles, C. L., Zha, H., Li, C., and Tsioutsiouliklis, K. 2004. Two supervised learning approaches for name disambiguation in author citations. In *JCDL*. 296–305.

Hassell, J., Aleman-Meza, B., and Arpinar, I. B. 2006. Ontology-driven automatic entity disambiguation in unstructured text. In *International Semantic Web Conference*. 44–57.

Ley, M. 2002. The DBLP computer science bibliography: Evolution, research issues, perspectives. In *SPIRE*. 1–10.

Mann, G. S. and Yarowsky, D. 2003. Unsupervised personal name disambiguation. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003*. Association for Computational Linguistics, Morristown, NJ, USA, 33–40.

Minkov, E., Cohen, W. W., and Ng, A. Y. 2006. Contextual search and name disambiguation in email using graphs. In *SIGIR*. 27–34.

Pedersen, T., Purandare, A., and Kulkarni, A. 2005. Name discrimination by clustering similar contexts. In *Proceedings of 6th International Conference on Computational Linguistics and Intelligent Text Processing*. 226–237.

Yarowsky, D. 1995. Unsupervised word sense disambiguation rivaling supervised methods. In *ACL*. 189–196.