Domain-Independent Entity Coreference for Linking Ontology Instances

Dezhao Song, Lehigh University
Jeff Heflin, Lehigh University

The objective of entity coreference is to determine if different mentions (e.g., person names, place names, database records, ontology instances, etc.) refer to the same real word object. Entity coreference algorithms can be used to detect duplicate database records and to determine if two Semantic Web instances represent the same underlying real word entity. The key issues in developing an entity coreference algorithm include how to locate context information and how to utilize the context appropriately. In this paper, we present a novel entity coreference algorithm for ontology instances. For scalability reasons, we select a neighborhood of each instance from an RDF graph. To determine the similarity between two instances, our algorithm computes the similarity between comparable property values in the neighborhood graphs. The similarity of distinct URIs and blank nodes is computed by comparing their outgoing links. In an attempt to reduce the impact of distant nodes on the final similarity measure, we explore a distance-based discounting approach. To provide the best possible domain-independent matches, we propose an approach to compute the discriminability of triples in order to assign weights to the context information. We evaluated our algorithm using different instance categories from five datasets. Our experiments show that the best results are achieved by including both our discounting and triple discrimination approaches.

Categories and Subject Descriptors: I.2.6 [**Learning**]: Knowledge acquisition

General Terms: Algorithms, Experimentation, Theory

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

ACM Reference Format:
Dezhao Song and Jeff Heflin. 2011. Domain-Independent Entity Coreference for Linking Ontology Instances. ACM J. Data Inform. Quality , , Article A (December 10), 29 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

The purpose of entity coreference[1] is to decide if different 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., 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

[1]Entity coreference is also referred to as deduplication [Elmagarmid et al. 2007] and entity disambiguation [Hassell et al. 2006].

Author's addresses: Dezhao Song and Jeff Heflin, Department of Computer Science and Engineering, Lehigh University. 19 Memorial Drive West, Bethlehem, PA 18015.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

appropriate way. On one hand, we need to collect context information for those different mentions. We can collect the context from the documents where the mentions occur. The Internet can be another source for finding context information. On the other hand, it is really important to utilize 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 [Bilenko et al. 2003]. Also, the collected data may come from heterogeneous sources and may not be complete. For instance, two news articles may describe different aspects of *James Henderson*. One article may mention the name and affiliation while the other one can include his name, date of birth, email address, etc. In addition, there may be noises in the data provided. For example, some date information is included in the context for *James Henderson* and it is treated as his date of birth; however, that date information could simply be the date of a social event that this *James Henderson* attended. An entity coreference algorithm needs to be able to deal with such problems and challenges.

Entity coreference in the Semantic Web [Berners-Lee et al. 2001] is used to detect equivalent ontology instances. In the 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 describing resources and their relationships and it is a W3C recommendation for representing information in the Web[2]. Two resources are connected via one or more predicates in the form of triple. A triple, <s, p, o>, consists of three parts: subject, predicate and object. The subject is an identifier (e.g., a URI) and the object can either be an identifier or a literal value, such as strings, numbers, dates, etc. A URI that takes the subject place in one triple can be the object in another; therefore, the triples themselves form a graph, the RDF graph. In an RDF graph, an ontology instance is represented by a URI; however, syntactically distinct URIs could actually represent the same real world entity. For instance, a person can have multiple URI identifiers in bibliographic databases such as DBLP [Ley 2002] and CiteSeer [Giles et al. 1998] but such URIs represent the same person; thus they are coreferent. In the Semantic Web, coreferent instances are linked to each other with the ***owl:sameAs*** predicate and then such coreference information can be further utilized by other aspects of Semantic Web related research, such as Semantic Web based question answering, information integration, etc.

There has been numerous research for linking ontology instances in the Semantic Web. Linked Data[3] [Bizer et al. 2009] is one of the leading efforts in this area. According to the latest statistics[4], there are currently 207 datasets (from various domains, e.g., media, geography, publications, etc.) in the Linked Open Data (LOD) Cloud with more than 28 billion triples and about 395 million links across different datasets. However, one problem of these existing *owl:sameAs* links is that they were generated with algorithms that are not precise enough. As recently reported by Halpin et al. [2010], only 50% ($\pm$ 21%) of the *owl:sameAs* links are correct. Therefore, there emerges the need to be able to automatically detect high quality *owl:sameAs* links between ontology instances from heterogeneous datasets.

In this paper, we present a novel entity coreference algorithm to detect coreferent ontology instances. In general, given a pair of instances of comparable classes, our algorithm tells if they are coreferent, i.e., refer to the same real world entity, such as the same person, publication, etc. In our algorithm, for a given instance, we find its neighborhood graph from the entire RDF graph through an expansion process and we end

[2]http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/
[3]http://linkeddata.org/
[4]http://www4.wiwiss.fu-berlin.de/lodcloud/state/

up having a set of paths starting from this instance and ending on another node in the RDF graph. Each path is composed of several triples. Next, we compute the discriminability of each triple, taking into account its predicate. Such discriminability is then discounted according to the triple's distance to the root node (the ontology instance). With such a distance-based discounting approach and the triple discriminability, we compute the weight of each path in the neighborhood graph (the context) of an ontology instance. Compared to systems that only include a subset of these features, our proposed algorithm achieves the best performance on four types of ontology instances from two distinct datasets. Furthermore, our system outperforms state-of-the-art systems when applied to three benchmark datasets for ontology instance matching. Finally, we examine the scalability of our proposed system and adopt one preselection technique to improve its scalability.

We organize the rest of the paper as following. Section 2 discusses related work. In Section 3, we describe how to find the context information for an ontology instance. Then in Section 4 we propose our triple discriminability learning scheme. We formally present our entity coreference algorithm in Section 5, and introduce a simple filtering technique to improve system scalability in Section 6. We show our evaluation results in Section 7 and conclude in Section 8.

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 [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. However, this approach relies on the in-document entity coreference system to give good results in order to collect decent context information. Gooi and Allan [2004] employ three models, the incremental/agglomerative vector space models and KL divergence, 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 because their experiments showed that the best results were achieved with this window size. Mann and Yarowsky [2003] utilize unsupervised clustering over a feature space to do coreference. They extract context information for each mention from web pages. The main difference is that they try to extract some more representative information from the web pages, such as biographical information, marriage, parent/child relationships and so on. 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 a set of 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]. Special types of Wikipedia pages (e.g., 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 (WSD) and duplicate record detection in databases are two closely related topics to entity coreference. A word can have multiple meanings while the task of WSD is to choose the most appropriate one based upon the word's context [Yarowsky 1995; Zhang and Heflin 2010], such as a piece of free text. Duplicate record detection is to detect duplicate tuples and remove redundancies from databases [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 forms of full name or first initial plus family name. Dong et al. [2005] proposed an entity coreference algorithm that exploits the relationships between different entities to improve system performance. They collectively resolve entities of multiple types via relational evidence propagation in dependency graphs. They applied the algorithm to multiple real world datasets and demonstrated its effectiveness. Kalashnikov and Mehrotra [2006] proposed RELDC (Relational-based Data Cleaning) to detect coreferent entities by analyzing entity relationships. The entities and their relationships are viewed as a graph where edges represent the relationships between entities. In order to scale to large graphs, certain optimization techniques are employed. They demonstrated the effectiveness of their algorithm on real world datasets in different domains: author, publication and movie. As a combination of collective and relational based approaches, Bhattacharya and Getoor [2007] developed a collective relational clustering algorithm that uses both attribute and relational information to detect coreferent entities and their algorithm shows certain improvement over comparison systems without their proposed techniques on three real world datasets. Different from the methods discussed above, Ioannou et al. [2010a] proposed a novel framework for entity coreference with uncertainty, trying to detect coreferent instances at query-time. In their system, possible coreferent relationships are stored together with the data with some probability; then a novel probabilistic query technique is employed for answering queries by considering such probabilities. Their system achieves better F1-scores when comparing to some offline entity coreference systems on two datasets.

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. [2006] proposed an ontology-driven disambiguation algorithm to match ontology instances created from the DBLP bibliography [Ley 2002] to mentions in DBWorld documents[5]. 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. Their algorithm achieves good performance: 97.1% in precision and 79.1% in recall. However, one problem is that the authors manually and selectively picked some triples of the instances to use, e.g., name and affiliation. The features (e.g., co-occurrence) were identified manually as well. For domains where it is difficult to obtain domain expertise, it may not be feasible to decide what information would be important and useful; and it would also be difficult to identify useful features for such domains.

Different from previous papers, other researchers developed algorithms for detecting coreferent ontology instances. Aswani et al. [2006] proposed an algorithm for match-

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

ing ontology instances. Their algorithm matches person instances from an ontology converted from the British Telecommunications (BT) digital library, containing 4,429 publications and 9,065 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. Similar to the paper by Hassel 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 returned web page is really a person's publication page or simply a page from DBLP where papers of distinct authors may co-exist. Both RiMOM [Li et al. 2009] and the system proposed by Song and Heflin [2010] adopt a similar idea to our proposed approach where each property in an ontology is assigned a weight. The core idea is that different properties may have quite different impact and thus for each property, a specific weight is assigned. Combining such property weights with string matching techniques, the similarity between two instances is computed. Compared to Song and Heflin's system, we consider all predicates in an expansion chain (to be described in Section 5.2) in the context of an instance, and we adopt a simple preselection technique to improve system scalability; also, in Section 7.6, we show that our algorithm outperforms RiMOM on three benchmark datasets.

ObjectCoref [Hu and Qu 2008] adopts a two step approach for detecting coreferent instances. First, it builds an initial set of coreferent instances via reasoning, i.e., by using the explicit semantics of *owl:sameAs, owl:InverseFunctionalProperty, owl:FunctionalProperty, owl:cardinality and owl:maxCardinality*. In a second step, ObjectCoref utilizes self-training [Zhou and Li 2010] to learn the discriminability of property pairs based on the coreferent instances used for training. The discriminability reflects how well each pair of properties can be used to determine whether two instances are coreferent or not. Similarly, LN2R [Saïs et al. 2009], CODI [Noessner et al. 2010] and ASMOV [Jean-Mary et al. 2009] also utilize a combination of reasoning based and string similarity or lexical similarity (e.g., WordNet [Miller 1995]) based techniques for matching ontology instances. One disadvantage of reasoning based approaches is that they highly depend on the correct expressions of the ontologies. For example, as reported by the developers of the ASMOV system, in some dataset, the *surname* property was declared to be functional while two instances with different object values of this property are said to be coreferent by the groundtruth. Ioannou et al. [2010b] developed a system that focuses on query time duplicate instance detection on RDF data. The key technique is to index RDF resources to enable efficient look-ups. By adaptively determining the query to the index, similar instances to the query instance can be efficiently retrieved.

Scalability is one important aspect of entity coreference algorithms. To scale entity coreference systems, one solution would be to efficiently determine if an instance pair could be coreferent by adopting some lightweight methods, which is generally called *indexing* or *blocking* [Christen 2011]. Best Five [Winkler 2005] is a set of manually identified rules for matching census data. However, developing such rules can be expensive, and domain expertise may not be available for various domains. Yan et al. [2007] proposed a modified sorted neighborhood algorithm, ASN, to learn dynamically sized blocks for each record. The records are sorted based upon a manually determined key. For a record r, it automatically finds the next N records that might be coreferent to r where N could vary for different records. They claimed that changing to different keys didn't affect the results but didn't report any data. Marlin [Bilenko and Mooney 2003] uses an unnormalized Jaccard similarity on the tokens between attributes by setting a threshold to 1, which is to find an identical token between the attributes. Although it was able to cover all true matches on some dataset, it only reduced the pairs to consider by 55.35%. BSL [Michelson and Knoblock 2006] adopted supervised learning to learn a blocking scheme, a disjunction of conjunctions of (method, attribute) pairs. It learns one conjunction each time to reduce as many pairs as possible; and by running iteratively, more conjunctions would be learned to increase coverage on true matches. However, supervised approaches require sufficient training data that may not always be available. As reported by Michelson and Knoblock [2006], when using only 10% of the groundtruth for training on some dataset, 4.68% fewer true matches were covered by BSL. In order to reduce the needs of training data, Cao et al. [2011] proposed a similar algorithm that utilizes both labeled and unlabeled data for learning the blocking scheme. Adaptive Filtering (AF) [Gu and Baxter 2004], All-Pairs [Bayardo et al. 2007], PP-Join(+) [Xiao et al. 2008b] and Ed-Join [Xiao et al. 2008a] are all inverted index based approaches. AF indexes the records on their bigrams. All-Pairs is a simple index based algorithm with certain optimization strategies. PP-Join(+) proposed a positional filtering principle that exploits the ordering of tokens in a record. Ed-Join employed filtering methods that explore the locations and contents of mismatching n-grams.

3. FIND NEIGHBORHOOD GRAPH

In this section, we address the problem of how to locate context information for ontology instances in an RDF graph. We collect paths in an RDF graph within a certain distance to an instance (the root node) that we do coreference on. We define a path as a sequence of nodes and predicates in an expansion chain in Equation 1:

$$path =< i, predicate[1], node[1], ..., predicate[n], node[n] > \tag{1}$$

where i is the instance, $node[i]$ ($i > 0$) is any other expanded node from the RDF graph and $predicate[i]$ is a predicate that connects two nodes in a path. We define a function $depth(path)$ that counts the number of predicates in a path. We will use the operator **+** for tuple concatenation, e.g., $< a, b > + < c, d >=< a, b, c, d >$.

Algorithm 1 formally presents this expansion process. Starting from an instance, we search for triples whose subject or object equals to the URI of this instance and record those expanded triples. With the expanded triples, if the objects or subjects are still URIs or blank nodes, we repeat this search or expansion process on them to get further expanded triples until we reach a depth limit or a literal value, whichever comes first. In an RDF graph, a blank node (or anonymous resource) is a node that is neither identified by a URI nor is a literal. Blank nodes have a node ID which is limited in scope to a serialization of a particular graph, i.e. the node $node[1]$ in one RDF graph does not represent the same node as a node named $node[1]$ in any other graph. At line 17, we use p^- to denote a predicate p when expanding from object to subject.

We implemented this expansion process as breadth-first search. In order to control the number of paths generated, we set a depth limit (the maximum number of predicates in a path) of 2. With this limit, we've discovered that it is sufficient to get enough context information. For example, in the RKB dataset[6], given a person instance, we can find its name, affiliation, and the URIs of this person's publication instances at depth 1; going further to depth 2, we will have the titles, dates and the URIs of the coauthors of these publications, etc.

With our expansion process, we end up having a set of paths for each ontology instance, starting from that instance and ending with a URI or a literal value. When ending on a blank node, we do not record that path because we cannot simply compare two blank nodes and see if they are identical. However, we rely on paths that go through blank nodes to get further literals and URIs before the stopping criteria is

[6]This is one of the datasets that we use for evaluation in Section 7.

Algorithm 1 Neighborhood(G, i), i is an ontology instance and G is an RDF graph; returns a set of paths for i collected from G

```
1.  $P \leftarrow all\ predicates\ in\ G$
2.  $path \leftarrow < i >$
3.  $expansion\_set \leftarrow \{path\},\ paths \leftarrow \emptyset$
4.  for all $path' \in expansion\_set$ do
5.    $last \leftarrow last\ node\ in\ path'$
6.    if $last\ is\ literal$ or ($last\ is\ URI$ and $depth(path') = depth\_limit$) then
7.      $paths \leftarrow paths \bigcup path'$
8.    else if $depth(path') < depth\_limit$ then
9.      $/ * Expand\ from\ subject\ to\ object * /$
10.     $triples \leftarrow \bigcup_{p \in P} \{t | t =< last, p, o > \wedge t \in G\}$
11.     for $t =< s, p, o > \in triples$ do
12.       $path\_new \leftarrow path' + < p,\ o >$
13.       $expansion\_set \leftarrow expansion\_set \bigcup path\_new$
14.     $/ * Expand\ from\ object\ to\ subject * /$
15.     $triples \leftarrow \bigcup_{p \in P} \{t | t =< s, p, last > \wedge t \in G\}$
16.     for $t =< s, p, o > \in triples$ do
17.       $path\_new \leftarrow path' + < p^-,\ s >$
18.       $expansion\_set \leftarrow expansion\_set \bigcup path\_new$
19.   $expansion\_set \leftarrow expansion\_set - path'$
20. return $paths$
```

met. As illustrated in Figure 1, starting from the root (an instance), we get to node 1, 2 and 3 by searching triples that use the root node as subject/object; then we reach node 4, 5, 6 and 7 by further expanding node 2, so on and so forth. We will explain P and F from Figure 1 in Section 4.4.

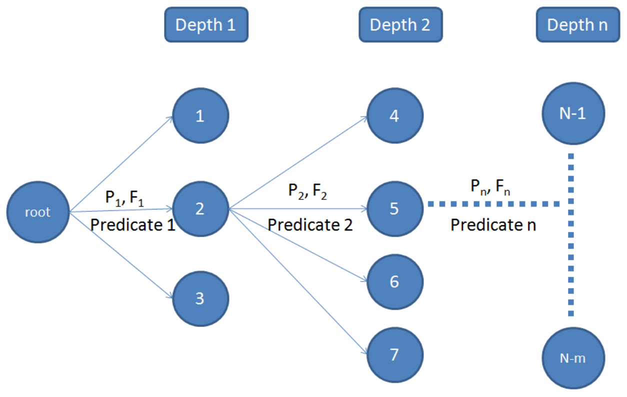


Fig. 1. Expansion Result

4. COMPUTE TRIPLE DISCRIMINABILITY

In this section, we will present our approach for learning the discriminability of RDF triples, explaining how we could utilize the collected context information appropriately.

Generally, each triple has its own importance, reflecting its possible level of discrimination to the ontology instance from which it originally comes from[7]. 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 compute the discriminabilities regardless of the domain of that dataset, such as academia domain or some others.

4.1. 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 discriminating it will be. Triples with different predicates, such as *has_publication_date* and *has_author*, could have different discriminabilities. Equations 2 and 3 show how we compute predicate discriminability:

$$Per_{p_i} = \frac{|set\ of\ distinct\ objects\ of\ p_i|}{|set\ of\ triples\ that\ use\ p_i|} \tag{2}$$

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 largest percentage value over all predicates as Per_{max}. We then normalize such values so that the most discriminating predicate has a discriminability of 1. The normalization is shown in equation 3:

$$P_{p_i} = \frac{Per_{p_i}}{Per_{max}} \tag{3}$$

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

Depending on what category of instances is being compared, a predicate may be used in the subject-to-object direction or reversely. 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 neighborhood triples in Section 3. Basically, when we do the expansion, we use a URI both as the subject and the object, so a predicate has different discriminabilities to the 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 $P^{-}_{p_i}$ respectively. Equations 2 and 3 compute predicate discriminabilities to the object direction. The discriminabilities to the subject direction can be computed in the same manner by replacing appropriate variables.

Here, we show how to calculate our predicate discriminability with two concrete examples from the RKB dataset. 6,313,274 triples use the predicate *has_author* (with domain of publication class and range of person class); among these triples, there are 3,986,181 distinct object values and 2,515,439 distinct subject values. From Equation 2, we have:

$$Per_{has\_author} = \frac{3,986,181}{6,313,274} = 0.63,\ \ Per^{-}_{has\_author} = \frac{2,515,439}{6,313,274} = 0.398 \tag{4}$$

[7]This is related to finding the neighborhood graph for an instance as introduced in Section 3.

Because the maximum percentage values to both directions (Per_{max} and Per^{-}_{max}) are both 1 in this dataset, based on Equation 3, we have:

$$P_{has\_author} = \frac{Per_{has\_author}}{Per_{max}} = \frac{0.63}{1} = 0.63, \tag{5}$$

$$P^{-}_{has\_author} = \frac{Per^{-}_{has\_author}}{Per^{-}_{max}} = \frac{0.398}{1} = 0.398 \tag{6}$$

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. Another example is the *has_publication_year* predicate (a datatype property). 2,973 triples use this predicate with 152 distinct object values. So, we have:

$$Per_{has\_publication\_year} = \frac{152}{2,973} = 0.05 \tag{7}$$

Based on Equation 3, we have:

$$P_{has\_publication\_year} = \frac{Per_{has\_pub\_year}}{Per_{max}} = \frac{0.05}{1} = 0.05 \tag{8}$$

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 coreference process. For example, if two publications 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 if 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 (having the same author) will give a better idea that these two publications be coreferent.

4.2. Predicate Discriminability Overestimation

Currently, when counting the size of the distinct object/subject value sets, we assume that if any two objects/subjects are syntactically 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.

Figure 2 gives an example of this situation. We have two predicates: PredicateA

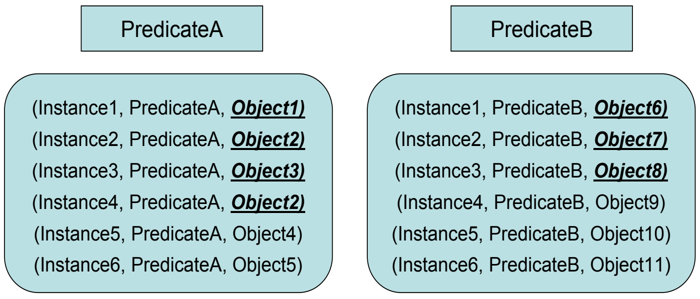

PredicateA
PredicateB
(Instance1, PredicateA, Object1)
(Instance2, PredicateA, Object2)
(Instance3, PredicateA, Object3)
(Instance4, PredicateA, Object2)
(Instance5, PredicateA, Object4)
(Instance6, PredicateA, Object5)
(Instance1, PredicateB, Object6)
(Instance2, PredicateB, Object7)
(Instance3, PredicateB, Object8)
(Instance4, PredicateB, Object9)
(Instance5, PredicateB, Object10)
(Instance6, PredicateB, Object11)

Fig. 2. Predicate Discriminability Overestimation

(P_A) and PredicateB (P_B) from the same dataset; each of them is used by six triples as listed in the two boxes respectively. Assuming Per_{max} is 1 for this dataset, based on Equations 2 and 3, to the object direction, we can calculate their discriminability:

$$P_{P_A} = \frac{Per_{P_A}}{Per_{max}} = \frac{5}{6} = 0.83,\ P_{P_B} = \frac{Per_{P_B}}{Per_{max}} = \frac{6}{6} = 1 \tag{9}$$

Now, if we assume the underlined objects are actually coreferent, the predicate discriminability of these two predicates will change to:

$$P_{P_A} = \frac{Per_{P_A}}{Per_{max}} = \frac{3}{6} = 0.5,\ P_{P_B} = \frac{Per_{P_B}}{Per_{max}} = \frac{4}{6} = 0.67 \tag{10}$$

In this case, we are overestimating the predicate discriminability for both predicates due to unknown coreferent instances. In our unsupervised method for learning predicate discriminability, it is difficult to consider such unknown coreferent information; therefore, we make the assumption that we overestimate the discriminability for every predicate by about the same proportion. From our current experiments (to be presented in Section 7.2), adopting our proposed predicate discriminability does significantly improve system performance compared to comparison systems (to be introduced in Section 7.1.3) that don't use it.

4.3. Missing Value

Currently, we don't consider missing values when calculating our predicate discriminability values. A version of Equation 2 that takes missing values into account is shown in Equation 11:

$$Per'_{p_i} = \frac{|set\ of\ distinct\ objects\ of\ p_i| + 1}{|triples\ that\ use\ p_i| + |instances\ that\ should\ use\ p_i\ but\ don't|} \tag{11}$$

where Per'_{p_i} represents the percentage value for predicate p_i. Compared to Equation 2, the additional "1" in the numerator represents *null value* and the second part ($Missing = |instances\ that\ should\ use\ p_i\ but\ don't|$) in the denominator is the number of instances that should have a value for predicate p_i but actually don't. The problem here is how to determine $Missing$. In an ontology, a predicate may not have its domain declared thus making it difficult to calculate $Missing$. Another option is to compute predicate discriminability with respect to each individual class as shown in Equation 12:

$$Per'_{(C,p_i)} = \frac{|set\ of\ distinct\ objects\ of\ p_i| + 1}{|triples\ that\ use\ p_i| + |instances\ of\ C\ that\ don't\ use\ p_i|} \tag{12}$$

where C is an ontology class. With this option, we compute the discriminability of all predicates for each individual class; thus a predicate may have different discriminabilities when paired with different classes. As shown in Figure 1, during expansion, we need to determine the discriminability of an edge (a predicate). However, the nodes to expand (e.g., the root node in Figure 1) may not have class types declared or may have multiple types; therefore one problem here is that we may not be able to decide which discriminability to use during the expansion process. For instance, in the RKB dataset, a person instance is also declared to be an *owl:Thing*, *Legal-Agent* and *Generic-Agent*. For predicate p_i, although we could compute its discriminability by Equation 12 with respect to each of these classes, it is difficult to decide which one to use when seeing an edge of p_i. In this paper, we didn't implement these two alternate options due to the problems discussed above when applying them generally to Semantic Web data. For future work, we will explore approaches for calculating predicate discriminability by appropriately taking into account missing values.

4.4. Weighted Neighborhood Graph

With triple discriminability and the context information, we assign each path in the neighborhood graph a weight, indicating its importance to the root node. The weights combine two elements, the learned discriminability and a discount value.

As previously shown in Figure 1, P_1 and P_2 represent the discriminabilities for the two triples ending on node 2 and 5 respectively. We also add another parameter, called factor, to each node, indicating how important a node is to its father node. For example, in Figure 1, F_1 is the factor of node 2 to the root node and its value is 1/3 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 triple discriminability, we adopt a distance based discounting approach to assign weights to paths in the neighborhood graph, as calculated by Equation 13:

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

where the function $depth$ counts the number of triples or predicates in a path; P_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.

5. ENTITY COREFERENCE ALGORITHM

With the weighted neighborhood graph, in this section, we formally present our entity coreference algorithm for detecting equivalent ontology instances.

5.1. Algorithm Design

Algorithm 2 presents the pseudo code of our entity coreference algorithm for ontology instances. In this description, a and b are two ontology instances; $path.node[last]$ returns the last 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 [Cohen et al. 2003].

The essential idea of our entity coreference algorithm is that we adopt the *bag-of-paths* approach to compare paths between ontology instances. Information retrieval systems often treat a document as a *bag-of-words* [Lewis 1998] where the text is treated as an unordered collection of words. Analogously, we treat an instance as a *bag-of-paths*. Through the expansion process (see Section 3), for an instance, we find a collection of paths for it without considering the ordering of the paths. For each path (m) of instance a, we compare its last node to that of every comparable path of instance b and choose the highest similarity score, denoted as $path\_score$. Also, we need to determine the weight of this path score. Here, when considering the weight, we take into account both the weight (W_m) of path m and the weight ($W_{n'}$) of the path n' of instance b that has the highest similarity to path m. Then we use the average of W_m and $W_{n'}$ as the path weight for path m. We then repeat the process for every path of instance a. 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 score between the two instances. The same process is repeated for all pairs of ontology instances of comparable cat-

Algorithm 2 Compare(N_a, N_b), N_a and N_b are the context for instances a and b collected with Algorithm 1 respectively; returns the similarity between 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 $m.node[last]\ is\ literal$ then
6.        $path\_score \leftarrow \max_{n' \in N_b, PathComparable(m,n')} Sim(m.node[last], n'.node[last])$
7.        /* path $n'$ has the highest score with m */
8.        $path\_weight \leftarrow (W_m + W_{n'})/2$
9.      else if $m.node[last]\ is\ URI$ then
10.       if $\exists path\ n' \in N_b$, $PathComparable(m, n')$, $m.node[last] = n'.node[last]$ 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.     $total\_score \leftarrow total\_score + path\_score * path\_weight$
15.     $total\_weight \leftarrow total\_weight + path\_weight$
16. return $\frac{total\_score}{total\_weight}$
```

egories, i.e., person-to-person and publication-to-publication. At line 16, the score (a float number) for a pair of instances is returned.

5.2. Path Comparability

As described, we compare the last nodes of instance a's paths to those of the comparable paths of instance b. One question here is how to determine if two paths are comparable. For example, the following two paths are not comparable (predicates in italic): (1). (personA, *attend_event*, eventA, *has_event_date*, 2010-06-01); (2). (personA, *has_publication*, article1, *has_publication_date*, 2010-06-01). Although the two last nodes are all date information and thus are comparable, they actually come from paths with different underlying semantics.

In our current approach, two paths are comparable if they satisfy the condition shown in Equation 14:

$$\begin{aligned} PathComparable(path_1, path_2) = true \equiv \\ (depth(path_1) = depth(path_2)) \ \wedge \\ (\forall i \in [1, depth(path_1)], \\ PredicateComparable(path_1.predicate[i], path_2.predicate[i]) = true) \end{aligned} \tag{14}$$

where $PathComparable$ and $PredicateComparable$ are two functions, telling if two paths or two predicates are comparable; $depth$ counts the number of predicates in a path; $path_1.predicate[i]$ and $path_2.predicate[i]$ return the ith predicate in two paths respectively.

There are several questions coming out from Equation 14. The first one is how to determine the comparability of two predicates, which will help to determine path comparability. In some situations, the comparability of predicates is not very clear. For instance, the 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. In such a circumstance, without letting such two predicates be comparable, we might miss some true matches while adding them in might hurt precision.

We say two predicates are comparable if the knowledge base (KB) entails that one is the subproperty of another (obviously, this means equivalent properties are also comparable). For our experiments, we created these mapping axioms manually. For example, the following two predicates are comparable: *citeseer:fullname* and *foaf:name*[8]. They are from different ontologies, but both represent person name. Please note that we also use entailment to determine if two classes are comparable. Therefore, we only compute the similarity of instance pairs of comparable classes. Ontology alignment [Euzenat 2004], a well studied topic in the Semantic Web, can help automatically determine predicate and class comparability across multiple ontologies, which is out of the scope of this paper.

Furthermore, when determining the comparability of two paths, we only care about the predicates in the paths but to ignore the intermediate nodes. The reason is that in two paths, intermediate nodes are URIs and distinct URIs do not necessarily represent distinct real world entities. Distinct URIs can actually represent two coreferent instances. Since we do not have the complete coreference knowledge between URIs, it is hard to involve intermediate nodes when determining path comparability. One possible solution is that we adopt an iterative entity coreference algorithm on all instances of comparable classes in a dataset and then take the coreference results of the current iteration into account in further iterations to help better determine path comparability. However, this will require certain level of system scalability because a dataset can have millions of instances.

5.3. Node Similarity

Given that two paths are comparable, if the two last nodes are two literal values, e.g., person names or publication titles, then we adopt the JaroWinklerTFIDF [Cohen et al. 2003] string matching algorithm to compute their similarity unless otherwise specified. In another situation, if the two nodes are both URIs in the RDF graph, then we will simply check 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 3.

5.4. Comparing All Nodes in Paths

One design choice we made is to compute the $path\_score$ between two comparable paths by only comparing their end nodes. Another option is to take into account those intermediate nodes when calculating $path\_score$ as shown in Algorithm 3.

From line 5 to 16, we calculate the $path\_score$ between two comparable paths as the average of all matching scores between nodes at corresponding positions; and we pick the maximum $path\_score$ between path m of instance a and all its comparable paths of instance b. Note, intermediate nodes must either be URIs or blank nodes. Blank nodes have a node ID which is limited in scope to a serialization of a particular graph, i.e. a blank node named $nodeA$ in one RDF graph does not represent the same node as a node with the same name in any other graph. Therefore, we do not compare two blank nodes or a blank node and a URI but only compare two URIs or two literals from line 8 to 14. At line 20, if two paths whose nodes are all URIs or blank nodes have a $path\_score$ of 0, meaning none of their nodes match, we do not update $path\_weight$. Because $path\_weight$ is initialized to be zero at line 4, $total\_score$ and $total\_weight$ will not be updated either at line 23 and 24. In this case, we are not applying any penalty to a path that ends on a URI and doesn't match any of its comparable paths from the other instance.

[8]FOAF stands for Friend of a Friend. More details are available at http://www.foaf-project.org/.

Algorithm 3 Compare_All_Nodes(N_a, N_b), N_a and N_b are the context of instances a and b collected with Algorithm 1; returns the similarity between 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.      $path\_weight \leftarrow 0, path\_score \leftarrow 0$
5.      for all $paths\ n' \in \{p | p \in N_b \wedge PathComparable(p, m)\}$ do
6.        $score \leftarrow 0, count \leftarrow 0$
7.        for $i <= depth(m)$ do
8.          if $m.node[i]\ is\ literal$ and $n'.node[i]\ is\ literal$ then
9.            $score \leftarrow score + Sim(m.node[i], n'.node[i])$
10.           $count \leftarrow count + 1$
11.         else if $m.node[i]\ is\ URI$ and $n'.node[i]\ is\ URI$ then
12.           if $m.node[i] = n'.node[i]$ then
13.             $score \leftarrow score + 1$
14.           $count \leftarrow count + 1$
15.       if $\frac{score}{count} > path\_score$ then
16.         $path\_score \leftarrow \frac{score}{count}$
17.     $n' \leftarrow path\ with\ the\ highest\ score\ compared\ to\ m$
18.     if $m.node[last]\ is\ literal$ then
19.       $path\_weight \leftarrow (W_m + W_{n'})/2$
20.     else if $m.node[last]\ is\ URI$ and $path\_score \neq 0$ then
21.       /* path $n'$ has at least one identical intermediate URI node with m */
22.       $path\_weight \leftarrow (W_m + W_{n'})/2$
23.     $total\_score \leftarrow total\_score + path\_score * path\_weight$
24.     $total\_weight \leftarrow total\_weight + path\_weight$
25. return $\frac{total\_score}{total\_weight}$
```

The rationale to only consider end nodes (as presented in Algorithm 2) is that the intermediate nodes from two paths could be syntactically different but are actually coreferent instances. So, if we apply penalties to mismatching intermediate nodes, it is possible for the system to miss some true matches. One potential advantage of matching intermediate nodes is that it may improve the system's precision because it is possible that the middle nodes are indeed different instances but the end nodes are coincidentally the same. We hypothesize that the improvement on precision cannot compensate the sacrificed recall when considering intermediate nodes and we will experimentally compare Algorithm 2 and Algorithm 3 in Section 7.3.

5.5. The Open World Problem

Another challenge that we face is that we cannot make a closed-world assumption, instead we need to deal with open-world [Russell and Norvig 2010]. We cannot assume something we don't know is false, everything we don't know is undefined. Within a Semantic Web 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 instances, only includes a certain portion of their information. For instance, these datasets might only contain some of these person instances' 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. As shown in Algorithm 2, if the last node of path m of instance a is a URI but it doesn't match any last node of comparable paths of instance b, we do not add any weight to *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 are no paths of instance b that are comparable to path m of instance a, 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, mismatches that are potentially caused by information incompleteness cannot simply be treated as real mismatches. We would like to investigate more sophisticated solutions to this problem in future work.

6. SCALABILITY

In order to improve system scalability, we tried a simple preselection technique. The general idea is that we want to quickly select those instance pairs that are likely to be coreferent by only comparing some identifying information of them. For example, for person instances, the names could be a good choice; while titles might be useful for performing preselection of publication instances. However, one question here is how to choose such identifying information in a domain-independent fashion. In other words, given different domains (different types of ontology instances), how would the system automatically pick the identifying information without human interferences?

In this paper, we choose to utilize the learned predicate discriminability as introduced in Section 4.1 to choose such identifying information. Recall that the predicate discriminability represents how discriminating a predicate is based upon the diversity of its object/subject value set. The more discriminating a predicate is, the more diverse its subject/object value set is and thus the more identifying its object/subject values are for the ontology instances. Therefore we could choose the most disambiguating predicate p_{max} as the identifying predicate and then use its object/subject values as the identifying information. Another question is that it is possible that not all instances have triples with p_{max}. For instance, in the collected RKB dataset, the predicate *has_email_address* has the highest predicate discriminability (to the object direction); while only 724 out of over three million person instances actually have such information.

In our approach, a predicate p_m is chosen as p_{max} if it satisfies Equation 15:

$$\arg\max_{p_m \in PC} P(p_m) \tag{15}$$

where p_m is a predicate; $P(p_m)$ computes the predicate discriminability of p_m; PC is a set of datatype properties that are used by all instances in a given dataset, which is computed in Equation 16:

$$PC = \{p | Range(p) = literal \land \forall i \in G, \exists j, t =< i, p, j >\in G\} \tag{16}$$

where i is an ontology instance of a particular type in dataset G; j represents any object value; p is a predicate and $Range(p)$ represents the range of p (i.e., literal or URI). These two equations mean that a predicate will be chosen as p_{max} if its range is literal, all instances in a given dataset use it and finally it needs to be more discriminating than other such predicates. Here, we require that p_{max} should have literal range because the underlying idea of this preselection is to filter by computing the similarity of some identifying information. For two URIs, we can only say if they are identical or not, which doesn't really give any similarity; furthermore, again, syntactically distinct URIs could be coreferent. We then pairwisely compare the object values of p_{max} of instances of comparable types in a dataset and select pairs whose object similarity is higher than a pre-defined threshold.

7. EVALUATION

In this section, we present how we collect and prepare our datasets, the evaluation metrics and methodology we use and the evaluation results by applying our entity coreference algorithm on different instance categories from datasets of various domains. Our system is implemented in Java and compiled with Java 1.6. All our experiments are conducted on a single Sun Workstation with one eight-core Intel Xeon 2.93GHz processor and 6GB memory.

7.1. Experiment Setup

7.1.1. Data Collection and Preparation. We evaluate our entity coreference algorithm on instances from the RKB and the SWAT datasets.

The RKB Dataset. The entire RKB dataset[9] [Glaser et al. 2008] consists of 54 RDF 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 of it: ACM, DBLP, CiteSeer, EPrints, IEEE, LAAS-CNRS, Newcastle, 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. The SWAT dataset[10] is another RDF dataset, consisting data parsed from the downloaded XML files of CiteSeer and DBLP. It has 26 million triples, 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 the SWAT dataset during transformation. Note that all *owl:sameAs* statements in both datasets are ignored while we collect the context for instances as described in Section 3. 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 both datasets. 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.

To increase the ambiguity of our test sets, we randomly picked 1,579 person and 2,102 publication instances from the RKB dataset and 1,010 person and 1,378 publication instances from the SWAT dataset through a filtering process. If the names/titles of two instances have a similarity score higher than 0.5 but are still said to be not coreferent based upon the groundtruth, we will put this pair of instances into an instance pool. Then we apply our algorithm on every pair of instances in each of the four test sets. We will describe how we achieve the goldstandard in section 7.1.2. There are 393,339, 78,444, 52,003 and 450,500 paths for our RKB publication, RKB person, SWAT person and SWAT publication 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 test sets.

7.1.2. Evaluation Metric and Methodology. Our algorithm does entity coreference on every pair of instances in the test sets and stores results in the form of (instanceA, instanceB, score). In our evaluations, we use the standard measures: precision, recall and F1-score

[9]http://www.rkbexplorer.com/data/
[10]http://swat.cse.lehigh.edu/resources/data/

as computed in Equations 17 and 18:

$$Precision_t = \frac{|correctly\ detected\ pairs|}{|totally\ detected\ pairs|},\ Recall_t = \frac{|correctly\ detected\ pairs|}{|true\ matches\ in\ test\ set|} \tag{17}$$

$$F1\text{-}Score_t = 2 * \frac{Precision_t * Recall_t}{Precision_t + Recall_t} \tag{18}$$

where t represents threshold in all three equations.

There are a few things to note about our evaluations. First of all, in this paper, we evaluate our algorithm on different instance categories from two different datasets. Groundtruth of the RKB dataset can be downloaded from their website. To verify the soundness of the RKB groundtruth, we manually verified 300 coreferent pairs of person instances and publication instances respectively, while there are 81,556 and 148,409 in total for person and publication respectively in our collected RKB dataset. For the SWAT dataset, we manually labeled the groundtruth.

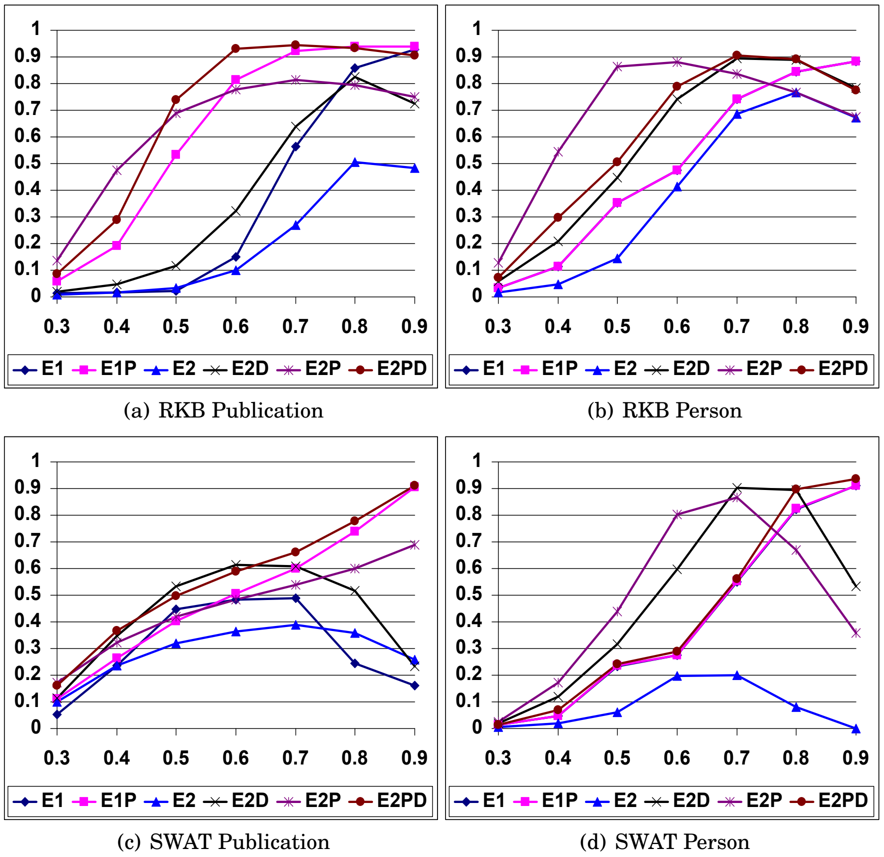
Furthermore, because the *owl:sameAs* predicate is transitive, i.e., if A is coreferent with B which is also coreferent with C, we will also have A and C are coreferent due to transitivity. In order to give the best correct evaluation results, we materialized all the coreferent pairs that can be achieved through reasoning on transitivity. Please note that we do not materialize reflexivity and symmetry.

Finally, in order to guarantee the completeness of the RKB groundtruth, we adopted a lazy or passive approach. We run our entity coreference algorithm on the two RKB test sets, and apply thresholds from 0.3 to 0.9 to evaluate the results based upon the groundtruth provided by RKB. Then we pick the comparison system (to be formally presented in section 7.1.3) that obtains the lowest precision, find out all the pairs that are detected by this system but are said to be not coreferent according to the groundtruth. For these *wrongly* detected pairs, we manually check each of them to see if any pair should be coreferent. We rely on the authors' DBLP and real homepages to perform such checks. Through this lazy-verification step, we were able to find 295 missing coreferent pairs for the RKB person test set. Mostly, RKB misses a coreferent pair of person instances because different subsets of RKB share little common publications for the two instances. Note that our 1,579 RKB person test set is generated after this lazy-verification.

7.1.3. Comparison Systems. In order to show the effectiveness of our proposed entity coreference algorithm, we compare our 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, representing predicate based triple discriminability), discount (D) which is implemented by using the factor. For example, the comparison system E2-D says it expands to depth 2, doesn't use triple discriminability but uses the discount of each expanded triple. So, for E2-D, Equation 13 will change to Equation 19:

$$W_{path} = \prod_{i=1}^{depth(path)} F_i \tag{19}$$

where path depth 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 discounting is from the factors. So, for this system, the weight of a path is computed with Equation



(a) RKB Publication (b) RKB Person

(c) SWAT Publication (d) SWAT Person

Fig. 3. F1-scores for RKB Publication, RKB Person, SWAT Publication and SWAT Person

20:

$$W_{path} = \prod_{i=1}^{depth(path)} P_i \tag{20}$$

where P_i is the predicate discriminability of a triple at depth i. For systems E1 and E2, the weight for every path in the neighborhood graph is set to 1. Our proposed algorithm, E2-P-D, then uses depth 2 expansion and adopts discounts and predicate based triple discriminability to form path weight.

7.2. Evaluate against Comparison Systems

In this section, we compare our proposed entity coreference algorithm to the comparison systems discussed in Section 7.1.3. Figures 3(a) to 3(d) show the F1-scores of our algorithm on the RKB publication, RKB person, SWAT publication and SWAT person datasets respectively. The x-axes are thresholds and the y-axes are percentage values, i.e., F1-scores. In these experiments, we adopted the JaroWinklerTFIDF string matching algorithm developed by Cohen et al. [2003]. From the F1-scores, we can see that our distance-based discounting entity coreference algorithm (E2-P-D) achieves the best performance on all four datasets. The F1-scores that our algorithm achieves for RKB publication, RKB person, SWAT publication and SWAT person are 94.4%, 90.6%, 91.0% and 93.8% at threshold 0.7, 0.7, 0.9 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 neighborhood graphs (contexts) in a broader range. But without discounts and discriminability, it is clearly worse than E1 for RKB publication, RKB person and SWAT person; for SWAT publication, although E2 has better results at thresholds 0.8 and 0.9, its best F1-score is much worse than that of E1. Such comparison shows that broader contexts can have negative impact if not appropriately managed.

By comparing E1-P to E1, it is not very clear that if only adding triple discriminability on adjacent triples gives better results. For the two publication datasets, adding such discriminability did lead to better results (except for threshold 0.9 and 0.5 for RKB publication and SWAT publication respectively); however, for the other two datasets, these two systems achieve very similar performance. Note that sometimes the curves of the two systems on the two person datasets are not very clear because they are overlapping.

Furthermore, E2-P is better than E2 for all datasets, though they achieve similar results for RKB person at thresholds 0.8 and 0.9. Different from the comparison between E1 and E1-P, adding triple discriminability to broader contexts significantly improved system performance. This shows the effectiveness of using a broader context with better management. The differences between E2 and E2-D show that by only applying factor discounts can also give us a significant improvement on all datasets. This verifies the effectiveness of discounting. For SWAT publication, although E2-D and E2 have similar results at thresholds 0.3 and 0.9, the best F1-score of E2-D is much higher than that of E2.

Last, our proposed algorithm, E2-P-D, is able to achieve the best performance for all datasets. Although precision and recall are not shown, E2-P-D has low precision at low thresholds, which can be partially because that we don't apply penalties on URI mismatches and missing literal information. But it is often able to improve as threshold rises, often topping out higher than any other system in the study. Most of the comparison systems using broader context information experience significant drops in recall at higher thresholds, but E2-P-D is less affected when applying high thresholds. Compared to E2-D, E2-P-D shows better results on RKB publication at all thresholds and on RKB person except at threshold 0.9; also, on SWAT publication, E2-P-D is clearly better than E2-D from thresholds 0.7 to 0.9. Again, this verifies the effectiveness of using predicate discriminability. E2-P-D shows significant improvement over E2-P for the two publication datasets (thresholds 0.5 to 0.9 for RKB publication and thresholds 0.4 to 0.9 for SWAT publication); for the two person datasets, it is not significantly better than E2-P but it is able to be on top at high thresholds. Such results demonstrate the effectiveness of combining triple discriminability and the discounting factor.

7.3. End Nodes Only vs. All Nodes in Paths

In Sections 5.1 and 5.4, we discussed two alternatives of computing the similarity between two comparable paths: only matching end nodes or matching all nodes of two paths. In this section, we experimentally compare these two alternatives on our four test sets as shown in Table I.

In general, only matching end nodes gives better recall while matching all nodes in paths leads to better precision. For RKB publication, RKB person and SWAT publication, the all-node version algorithm has better F1-scores at low thresholds due to its higher precision and comparably good recall; however, for higher thresholds, the end-node version algorithm wins out because of its less affected recall and improved precision. The end-node version algorithm has the best F1-scores for these three datasets.

Table I. Matching End Nodes vs. Matching All Nodes

Dataset	Metric	System	Threshold				
			0.5	0.6	0.7	0.8	0.9
RKB Publication	Precision	End-Node	58.75	87.74	95.42	97.71	**99.71**
		All-Node	**64.46**	**89.68**	**96.41**	**98.44**	99.41
	Recall	End-Node	**100**	**99.28**	**93.42**	**89.29**	**82.72**
		All-Node	99.58	95.63	88.28	68.06	30.02
	F1-score	End-Node	74.02	**93.15**	**94.41**	**93.31**	**90.42**
		All-Node	**78.26**	92.56	92.16	80.48	46.12
RKB Person	Precision	End-Node	34.09	67.18	88.40	97.89	**99.21**
		All-Node	**43.18**	**75.76**	**89.39**	**98.06**	97.31
	Recall	End-Node	**98.32**	**95.88**	**92.94**	**81.92**	**63.75**
		All-Node	96.38	91.76	78.64	59.46	21.28
	F1-score	End-Node	50.63	79.00	**90.61**	**89.19**	**77.62**
		All-Node	**59.64**	**83.00**	83.67	74.03	34.92
SWAT Publication	Precision	End-Node	33.16	41.72	49.51	63.79	**84.13**
		All-Node	**34.67**	**44.35**	**55.23**	**69.15**	80.72
	Recall	End-Node	**100**	**100**	**99.90**	**99.69**	**99.07**
		All-Node	100	99.48	97.09	81.00	42.16
	F1-score	End-Node	49.81	58.88	66.21	**77.80**	**90.99**
		All-Node	**51.48**	**61.35**	**70.41**	74.61	55.39
SWAT Person	Precision	End-Node	13.74	16.93	39.27	83.64	91.09
		All-Node	**13.82**	**17.75**	**44.69**	**85.88**	**92.98**
	Recall	End-Node	**97.85**	97.42	97.42	96.57	96.57
		All-Node	97.42	97.42	97.42	96.57	96.57
	F1-score	End-Node	24.10	28.84	55.98	89.64	93.75
		All-Node	**24.21**	**30.03**	**61.27**	**90.91**	**94.74**

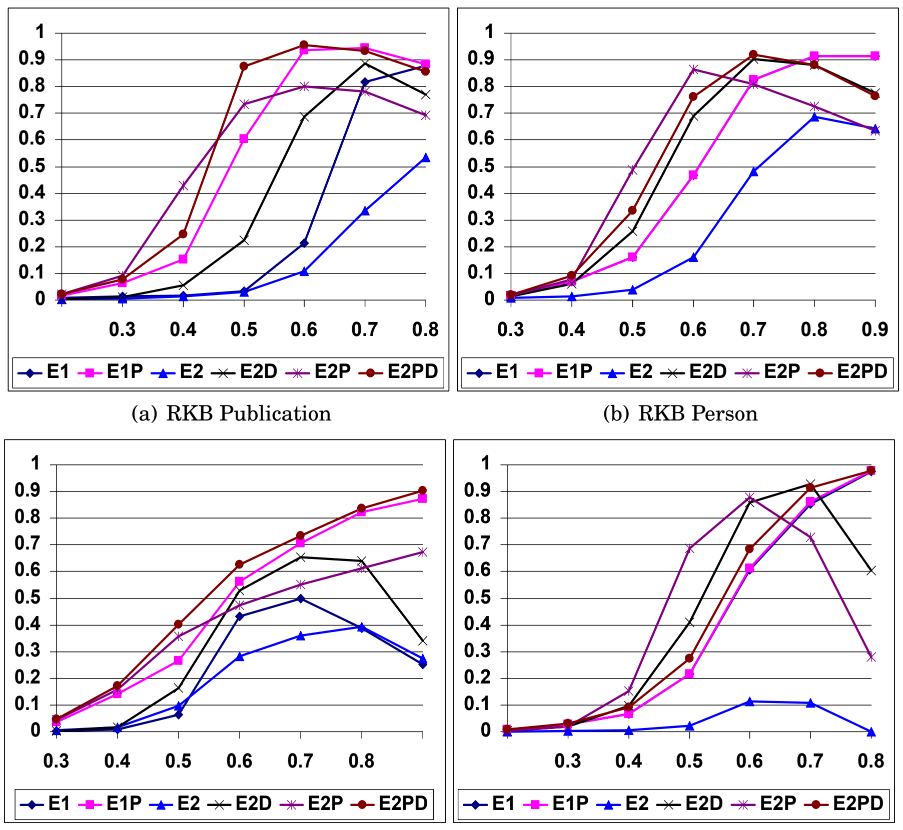
Note: We bold the higher scores that a system achieves than the other for each threshold on a dataset and also underline the best F1-scores for all thresholds for each dataset.

For SWAT person, all-node has the best F1-score because it has better precision than end-node and its recall doesn't get affected when applying high thresholds. Although end-node is not as good as all-node on SWAT person, only about 1% difference in their best F1-scores was observed; the best F1-scores of end-node are 1.85%, 6.94% and 16.38% higher than those of all-node on RKB publication, RKB person and SWAT publication respectively.

7.4. Robustness of Similarity Computations

In our system, we heavily rely on string matching to obtain the similarity between each pair of paths and thus the similarity between two instances. In the results presented in Section 7.2, we adopted the JaroWinklerTFIDF string matching algorithm from the secondstring package [Cohen et al. 2003] and we achieved good results. However, string matching algorithms should not be the dominating factor in our system. In this section, we show that our proposed system, E2-P-D, is robust in that it is able to achieve the best performance regardless of the chosen string matching algorithm. Figures 4(a) to 5(d) show the F1-scores for RKB publication, RKB person, SWAT publication and SWAT person datasets by adopting Edit distance [Levenshtein 1966] and Jaccard [Jaccard 1901] similarity measure.

From the results, we can see that our system is not subject to different string matching algorithms. It is true that the achieved F1-scores may vary, the shape of the curves may drift left or right and some other systems are able to achieve equally good results. However, E2-P-D was able to obtain the best F1-scores for all datasets with Edit distance and Jaccard similarity while none of the other comparison systems was able to achieve such results.



(a) RKB Publication (b) RKB Person

(c) SWAT Publication (d) SWAT Person

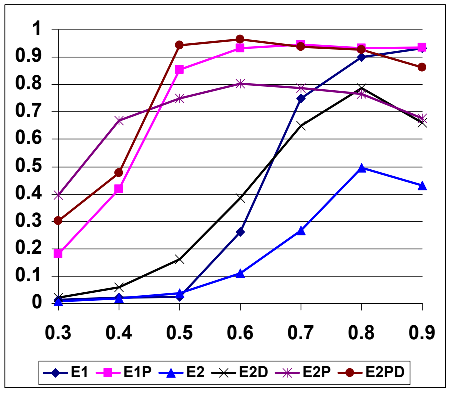
Fig. 4. F1-Scores for Edit Distance String Matching

7.5. System Scalability

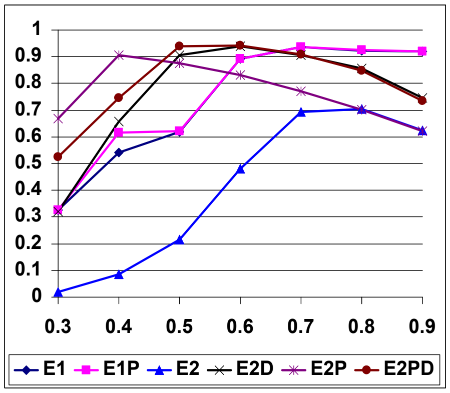
In this section, we examine the scalability of our proposed system (E2-P-D) and employ the preselection technique introduced in Section 6 to improve its scalability. For all experiments in this section, we use Jaccard similarity for string matching.

First of all, Figure 6(a) shows the runtime by applying our algorithm to 2,000 to 20,000 instances. The y-axis shows how many seconds were needed for the algorithm to finish and we use a logarithmic scale with base 10. From the results of RKB person naive, RKB publication naive, SWAT person naive and SWAT publication naive, we can see that E2-P-D doesn't scale well since it simply conducts a naive pairwise comparison on all pairs of instances in a given dataset.

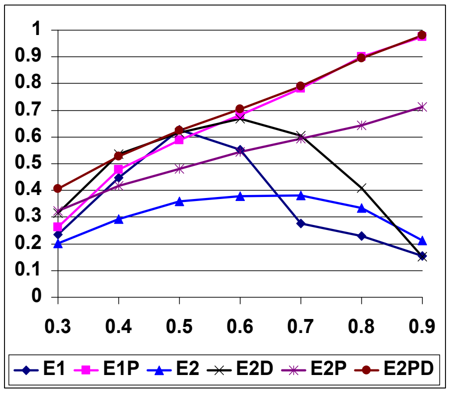
By applying the preselection technique, we re-run our scalability test and the four preselection curves in Figure 6(a) show that the system scales better after preselection. Please note that the runtime for the improved system includes both the time for preselection and entity coreference. Since we are at our early stage in exploring the scalability issue of entity coreference systems, our current approach is simple compared to some existing research [Michelson and Knoblock 2006; Yan et al. 2007; Cao et al. 2011]; however, our approach is unsupervised that it does not require any prelabeled groundtruth for preselection; also, our approach is domain-independent in the sense that it is based upon the predicate discriminability that is computed in a domain-independent and automatic manner.



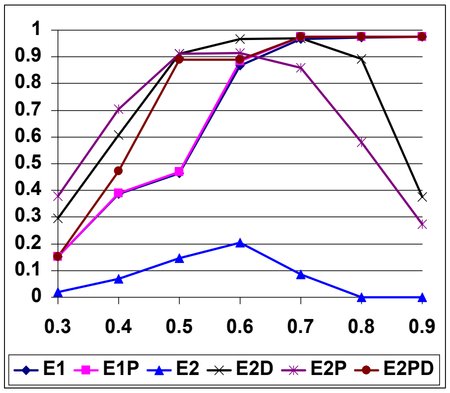
(a) RKB Publication



(b) RKB Person

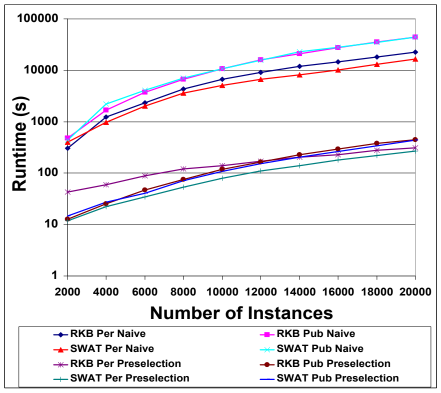


(c) SWAT Publication

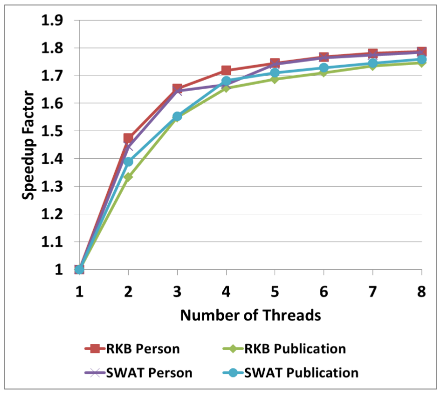


(d) SWAT Person

Fig. 5. F1-Scores for Jaccard Distance String Matching



(a) System Scalability



(b) Speedup Factor with Different Number of Threads

Fig. 6. System Scalability

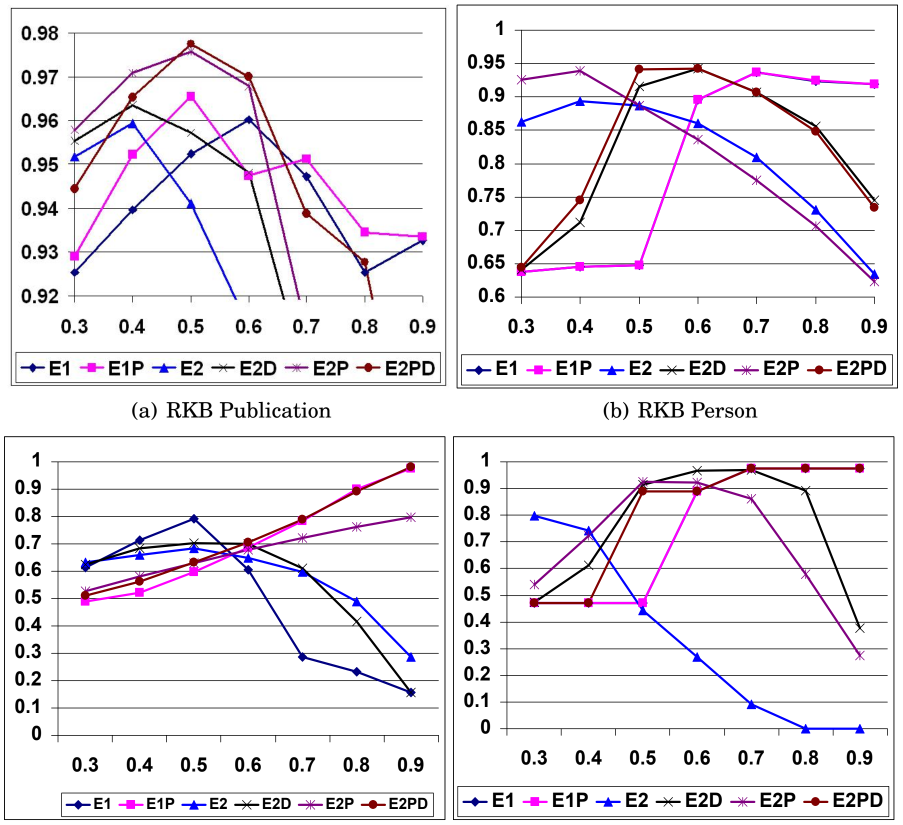
Table II. Preselection Results

Dataset	Recall	Number of Selected Pairs
RKB Person	0.965	2800
RKB Publication	1	2085
SWAT Person	0.974	1950
SWAT Publication	1	3081

In this scalability test, we parallelized our algorithm on 5 threads. Suppose we have N instances, the number of needed comparisons will be $M = \frac{N*(N-1)}{2}$. We equally divide this M comparisons to the five threads. We put the context information of the N instances into memory and all threads fetch the context information from this in-memory data structure. Figure 6(b) shows the speedup factor by distributing the M comparisons to one to eight threads. The speedup factor is computed as $Speedup\_Factor_k = \frac{T_1}{T_k}$, where k is the number of threads and T_k is the runtime by deploying k threads. Except for SWAT Person, the other three datasets didn't get significant improvement on this speedup factor by switching from 4 to 5 threads. With more than 5 threads, all datasets continue to achieve higher speedup factors but with diminishing returns.

We also did experiments to verify the effectiveness of our preselection approach by examining how much recall it is able to achieve and how many pairs it selects. The idea is that the preselection technique should give a decent level of recall so that the overall performance of the entire system will not be affected much; meanwhile, it should be selective enough in that it cannot select too many pairs. As introduced in Section 7.1.2, because of limited available groundtruth information, we still evaluate this preselection technique on the four test sets introduced in Section 7.1.1: 1,579 RKB person, 2,102 RKB publication, 1,010 SWAT person and 1,378 SWAT publication instances. Table II shows the number of selected pairs and recall of the applied preselection technique. We set the threshold to be 0.4, meaning that if the similarity between the identifying information (e.g., names or titles) of two instances is higher than 0.4, they are preselected. The reason for setting a low threshold here is to ensure high recall from the selected pairs. People names can be represented differently: first name + last name, first initial + last name, etc.; there could also be misspellings. For example, in our RKB Person test set, two names *Carlos J. Pereira de Lu* and *Carlos José Pereira de Lucena* are very similar but only have a similarity of 0.43. From the results, we see that our method achieves a decent recall; although we adopted a relatively low threshold, our method significantly reduces the total number of instance pairs that need to be compared with the expensive E2-P-D algorithm.

We should also check the F1-scores by applying E2-P-D and other comparison systems only to the preselected pairs. Figures 7(a) to 7(d) demonstrate the results after we plug in this preselection technique into the entire entity coreference process. From the results, we can see that our proposed system, E2-P-D, still achieves the best F1-scores for all datasets and other comparison systems generally achieve better F1-scores. Such improvement primarily comes from better precision. The reason is that the preselection process helped filtering out some false positives. For instance, without preselection, two person instances could have been said to be coreferent by comparing their names, publications, etc. while only comparing their names may not even make them to be preselected. The recall was not affected much and it could be the reason that the missing groundtruth cannot be detected by those systems even without preselection. Although E2-P-D still achieves the best performance for all datasets, other systems are able to obtain equally (SWAT person and publication) or comparably good results (RKB Publication and Person) to E2-P-D. One problem with our approach is that currently we do pairwise comparison based preselection; therefore, the preselection technique itself may not scale for larger datasets (with millions of instances). Even only comput-



(a) RKB Publication (b) RKB Person

(c) SWAT Publication (d) SWAT Person

Fig. 7. F1-Scores with Preselection and Using Jaccard Distance

ing the similarity of the names of millions of instances could still be time-consuming. For future work, we will explore how to locate candidate pairs more efficiently.

7.6. Compare to State-of-the-art Systems

To further demonstrate the capability of our proposed entity coreference algorithm, we compare E2-P-D to other systems that participated the OAEI2010 (Ontology Alignment Evaluation Initiative 2010) Campaign[11] [Euzenat et al. 2010] on the Person-Restaurant (PR) benchmark designed for ontology instance matching. We compare to RiMOM [Li et al. 2009], ObjectCoref [Hu and Qu 2008], LN2R [Saïs et al. 2009], CODI [Noessner et al. 2010], and ASMOV/ASMOV_D [Jean-Mary et al. 2009] on the three datasets of PR: Person1, Person2 and Restaurant. Person1 and Person2 are two synthetic datasets where coreferent records are generated by modifying the original records; Restaurant is a real-world dataset, matching instances describing restaurants from Fodors (331 instances) to Zagat (533 instances) with 112 duplicates[12]. Furthermore, we compare to the entity coreference system proposed by Dey et al. [2011] on a synthetic census dataset generated with FEBRL [Christen 2008]. We compare to these systems by referencing their published results as shown in Table III.

[11]http://oaei.ontologymatching.org/2010/

[12]For full details of these datasets, please refer to the OAEI2010 report [Euzenat et al. 2010].

Table III. Comparing to Other Systems

Dataset	System	$Precision(\%)$	$Recall(\%)$	$F1(\%)$
Person1	E2-P-D	**100**	**100**	**100**
	RiMOM [Li et al. 2009]	**100**	**100**	**100**
	ObjectCoref [Hu and Qu 2008]	**100**	99.8	99.9
	LN2R [Saïs et al. 2009]	**100**	**100**	**100**
	CODI [Noessner et al. 2010]	87	96	91
	ASMOV_D [Jean-Mary et al. 2009]	**100**	76.6	87
	ASMOV [Jean-Mary et al. 2009]	**100**	**100**	**100**
Person2	E2-P-D	98.52	**99.75**	**99.13**
	RiMOM [Li et al. 2009]	95.2	99	97.1
	ObjectCoref [Hu and Qu 2008]	**100**	90	94.7
	LN2R [Saïs et al. 2009]	99.4	88.25	93
	CODI [Noessner et al. 2010]	83	22	36
	ASMOV_D [Jean-Mary et al. 2009]	98.2	13.5	24
	ASMOV [Jean-Mary et al. 2009]	70.1	23.5	35
Restaurant	E2-P-D	74.58	98.88	**85.02**
	RiMOM [Li et al. 2009]	**86**	76.8	81.1
	ObjectCoref [Hu and Qu 2008]	58	**100**	73
	LN2R [Saïs et al. 2009]	75.67	75	75
	CODI [Noessner et al. 2010]	71	72	72
	ASMOV_D [Jean-Mary et al. 2009]	69.6	69.6	69.6
	ASMOV [Jean-Mary et al. 2009]	69.6	69.6	69.6
Census	E2-P-D	**100**	**99.10**	**99.55**
	Dey et al. [Dey et al. 2011]	99	98	98.50

Note: We bold the best scores that one or more systems achieve for each of the four datasets.

On Person1 and Person2, our system achieves the best F1-score on both datasets. Although RiMOM, ObjectCoref, LN2R and ASMOV also achieve good results on Person1, their performances drop significantly on Person2. This could be due to the difference between how coreferent instances were generated in these two datasets. For Person1, each original instance has at most one coreferent instance with a maximum of 1 modification per coreferent instance and a maximum of 1 modification per attribute of the original instance. Person 2 is created as Person 1, but with a maximum of 3 modifications per attribute, and a maximum of 10 modifications per instance. On the Restaurant dataset, both RiMOM and LN2R achieve better precision than our algorithm, but their recall is much lower than ours. E2-P-D has better precision than ObjectCoref while is only slightly worse on recall. Although E2-P-D is not the best in either precision or recall, it has significantly better F1-score than the other systems. Finally, on the census dataset, our algorithm achieves better performance than that of Dey et al. on all three metrics.

7.7. Discussion

The results show certain advantages of our approach; however, there are a few points to discuss. First, as we apply higher thresholds, recall generally goes down for all systems. As we described in Section 5.5, 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 score for them. The intuition behind such merging is to let the context to evolve to be more comprehensive. By employing an iterative entity coreference algorithm, we continue to compare the merged contexts, and therefore could potentially reduce the chance to miss a true match caused by information incompleteness from heterogeneous data sources. However we need to be very careful about doing such merges, since it is easy to add noise to the data if the standard for merging is not appropriately set. For example, two person instances (a and b) with names *James Smith* and *John Smith* from the same institution co-authored a paper. In this case, the similarity between instances a and b could be high because they share the same last name, work for the same institution and furthermore have the same publication whose title, publication date, venue, etc. information might be available. If we decide to merge the contexts of these two instances to make a combined instance c, the context of c actually contains information of two distinct instances and thus is noisy. In the next iteration, when we compute the similarity between c and other instances (e.g., person d), instances that shouldn't have been merged if individually compared to a and b could be merged due to some matchings provided by the noisy context of c. Iteratively, we could then have contexts that are more and more noisy which could ultimately affect precision.

Another problem is that, 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.

8. CONCLUSION AND FUTURE WORK

In this paper, we propose an entity coreference algorithm for detecting equivalent Semantic Web instances. Our algorithm finds a neighborhood graph of an ontology instance as its context information. With our triple discriminability learning scheme and the 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 score between a pair of ontology instances. We demonstrate the effectiveness of our entity coreference algorithm on test sets from different domains: author, publication, census and restaurant. Furthermore, although our algorithm heavily utilizes string matching, we show that it is not subject to different string matching algorithms. We also tested and improved the scalability of our system with a preselection technique that effectively reduces the total computation time and still allows our system to achieve the best F1-scores. Finally, we show that our system outperforms some state-of-the-art entity coreference systems.

For future work, we plan to explore iterative algorithms in order to apply appropriate penalties on URI mismatches and perform merging of instance contexts. Also, we will continue to investigate other techniques to improve the scalability of our system. Finally, we are interested in graph-based matching algorithms because, essentially, the context of 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.

BAYARDO, R. J., MA, Y., AND SRIKANT, R. 2007. Scaling up all pairs similarity search. In *Proceedings of the 16th International Conference on World Wide Web (WWW)*. 131–140.

BEKKERMAN, R. AND MCCALLUM, A. 2005. Disambiguating web appearances of people in a social network. In *WWW*. 463–470.

BERNERS-LEE, T., HENDLER, J., AND LASSILA, O. 2001. The semantic web. *Scientific American 284,* 5, 34–43.

BHATTACHARYA, I. AND GETOOR, L. 2007. Collective entity resolution in relational data. *TKDD 1,* 1.

BILENKO, M. AND MOONEY, R. J. 2003. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. KDD '03. ACM, New York, NY, USA, 39–48.

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.

BIZER, C., HEATH, T., AND BERNERS-LEE, T. 2009. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst. 5,* 3, 1–22.

BUNESCU, R. C. AND PASCA, M. 2006. Using encyclopedic knowledge for named entity disambiguation. In *EACL*.

CAO, Y., CHEN, Z., ZHU, J., YUE, P., LIN, C.-Y., AND YU, Y. 2011. Leveraging unlabeled data to scale blocking for record linkage. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*.

CHRISTEN, P. 2008. Febrl -: an open source data cleaning, deduplication and record linkage system with a graphical user interface. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. KDD '08. ACM, New York, NY, USA, 1065–1068.

CHRISTEN, P. 2011. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Transactions on Knowledge and Data Engineering 99,* PrePrints.

COHEN, W. W., RAVIKUMAR, P. D., AND FIENBERG, S. E. 2003. A comparison of string distance metrics for name-matching tasks. In *IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03)*. 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, 3–26.

DEY, D., MOOKERJEE, V. S., AND LIU, D. 2011. Efficient techniques for online record linkage. *IEEE Trans. Knowl. Data Eng. 23,* 3, 373–387.

DONG, X., HALEVY, A., AND MADHAVAN, J. 2005. Reference reconciliation in complex information spaces. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. SIGMOD '05. ACM, New York, NY, USA, 85–96.

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.

EUZENAT, J. 2004. An api for ontology alignment. In *International Semantic Web Conference*. 698–712.

EUZENAT, J., FERRARA, A., MEILICKE, C., NIKOLOV, A., PANE, J., SCHARFFE, F., SHVAIKO, P., STUCKENSCHMIDT, H., SVB-ZAMAZAL, O., SVTEK, V., AND TROJAHN DOS SANTOS, C. 2010. Results of the ontology alignment evaluation initiative 2010. In *Proceedings of the 4th International Workshop on Ontology Matching (OM) collocated with the 9th International Semantic Web Conference (ISWC)*.

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

GILES, C. L., BOLLACKER, K. D., AND LAWRENCE, S. 1998. Citeseer: an automatic citation indexing system. In *Proceedings of the third ACM conference on Digital libraries*. DL '98. ACM, New York, NY, USA, 89–98.

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.

GU, L. AND BAXTER, R. A. 2004. Adaptive filtering for efficient record linkage. In *Proceedings of the Fourth SIAM International Conference on Data Mining*.

HALPIN, H., HAYES, P. J., MCCUSKER, J. P., MCGUINNESS, D. L., AND THOMPSON, H. S. 2010. When owl: sameas isn't the same: An analysis of identity in linked data. In *9th International Semantic Web Conference (ISWC)*. 305–320.

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.

HU, W. AND QU, Y. 2008. Falcon-ao: A practical ontology matching system. *Journal of Web Semantics 6,* 3, 237–239.

IOANNOU, E., NEJDL, W., NIEDERÉE, C., AND VELEGRAKIS, Y. 2010a. On-the-fly entity-aware query processing in the presence of linkage. *PVLDB 3,* 1, 429–438.

IOANNOU, E., PAPAPETROU, O., SKOUTAS, D., AND NEJDL, W. 2010b. Efficient semantic-aware detection of near duplicate resources. In *The Semantic Web: Research and Applications, 7th Extended Semantic Web Conference (ESWC)*. Springer, 136–150.

JACCARD, P. 1901. Distribution de la flore alpine dans le bassin des dranses et dans quelques régions voisines. *Bulletin de la Société Vaudoise des Sciences Naturelles 37*, 241–272.

JEAN-MARY, Y. R., SHIRONOSHITA, E. P., AND KABUKA, M. R. 2009. Ontology matching with semantic verification. *Journal of Web Semantics 7*, 235–251.

KALASHNIKOV, D. V. AND MEHROTRA, S. 2006. Domain-independent data cleaning via analysis of entity-relationship graph. *ACM Trans. Database Syst. 31,* 2, 716–767.

LEVENSHTEIN, V. I. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady 10,* 8, 707–710.

LEWIS, D. D. 1998. Naive (bayes) at forty: The independence assumption in information retrieval. In *Proceedings of the 10th European Conference on Machine Learning*. Springer-Verlag, London, UK, 4–15.

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

LI, J., TANG, J., LI, Y., AND LUO, Q. 2009. RiMOM: A dynamic multistrategy ontology alignment framework. *Knowledge and Data Engineering, IEEE Transactions on 21,* 8, 1218–1232.

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.

MICHELSON, M. AND KNOBLOCK, C. A. 2006. Learning blocking schemes for record linkage. In *The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, AAAI*.

MILLER, G. A. 1995. Wordnet: A lexical database for english. *Commun. ACM 38,* 11, 39–41.

MINKOV, E., COHEN, W. W., AND NG, A. Y. 2006. Contextual search and name disambiguation in email using graphs. In *SIGIR*. 27–34.

NOESSNER, J., NIEPERT, M., MEILICKE, C., AND STUCKENSCHMIDT, H. 2010. Leveraging terminological structure for object reconciliation. In *The Semantic Web: Research and Applications, 7th Extended Semantic Web Conference (ESWC)*. 334–348.

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.

RUSSELL, S. J. AND NORVIG, P. 2010. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education.

SAÏS, F., PERNELLE, N., AND ROUSSET, M.-C. 2009. Combining a logical and a numerical method for data reconciliation. *Journal on Data Semantics XII 12*, 66–94.

SONG, D. AND HEFLIN, J. 2010. Domain-independent entity coreference in rdf graphs. In *Proceedings of the 19th ACM international conference on Information and knowledge management*. CIKM '10. ACM, New York, NY, USA, 1821–1824.

WINKLER, W. E. 2005. Approximate string comparator search strategies for very large administrative lists. Tech. rep., Statistical Research Division, U.S. Census Bureau.

XIAO, C., WANG, W., AND LIN, X. 2008a. Ed-join: an efficient algorithm for similarity joins with edit distance constraints. *Proc. VLDB Endow. 1,* 1, 933–944.

XIAO, C., WANG, W., LIN, X., AND YU, J. X. 2008b. Efficient similarity joins for near duplicate detection. In *Proceeding of the 17th international conference on World Wide Web*. WWW '08. ACM, New York, NY, USA, 131–140.

YAN, S., LEE, D., KAN, M.-Y., AND GILES, L. C. 2007. Adaptive sorted neighborhood methods for efficient record linkage. In *Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries*. JCDL '07. ACM, New York, NY, USA, 185–194.

YAROWSKY, D. 1995. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*. ACL '95. Association for Computational Linguistics, Stroudsburg, PA, USA, 189–196.

ZHANG, X. AND HEFLIN, J. 2010. Calculating word sense probability distributions for semantic web applications. In *Proceedings of the 2010 IEEE Fourth International Conference on Semantic Computing*. ICSC '10. IEEE Computer Society, Washington, DC, USA, 470–477.

ZHOU, Z.-H. AND LI, M. 2010. Semi-supervised learning by disagreement. *Knowl. Inf. Syst. 24*, 415–439.

Received December 2010; revised ; accepted