Accuracy vs. Speed: Scalable Entity Coreference on the Semantic Web with On-the-Fly Pruning

Dezhao Song
Department of Computer Science and Engineering
Lehigh University
Bethlehem, PA 18015, USA
des308@lehigh.edu

Jeff Heflin
Department of Computer Science and Engineering
Lehigh University
Bethlehem, PA 18015, USA
heflin@cse.lehigh.edu

***Abstract*—One challenge for the Semantic Web is to scalably establish high quality *owl:sameAs* links between coreferent ontology instances in different data sources; traditional approaches that exhaustively compare every pair of instances do not scale well to large datasets. In this paper, we propose a pruning-based algorithm for reducing the complexity of entity coreference. First, we discard candidate pairs of instances that are not sufficiently similar to the same pool of other instances. A sigmoid function based thresholding method is proposed to automatically adjust the threshold for such commonality on-the-fly. In our prior work, each instance is associated with a context graph consisting of neighboring RDF nodes. In this paper, we speed up the comparison for a single pair of instances by pruning insignificant context in the graph; this is accomplished by evaluating its potential contribution to the final similarity measure. We evaluate our system on three Semantic Web instance categories. We verify the effectiveness of our thresholding and context pruning methods by comparing to nine state-of-the-art systems. We show that our algorithm frequently outperforms those systems with a runtime speedup factor of 18 to 24 while maintaining competitive F1-scores. For datasets of up to 1 million instances, this translates to as much as 370 hours improvement in runtime.**

***Keywords*-Semantic Web, Entity Coreference, Linked Data, Scalability, Pruning**

I. Introduction

The purpose of entity coreference is to decide which identifiers (e.g., person names, locations, etc.) refer to the same real world entity. This is an essential task when integrating data from multiple sources. Entity coreference in the Semantic Web is used to detect equivalent (coreferent) ontology instances. Minimally, 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 their relationships (class hierarchies and predicates). RDF is a graph based data model for describing resources and their relationships in the Web. The source s of an edge, called its subject in RDF, is an identifier (i.e., a URI). The target o of an edge, called its object in RDF, can be either an identifier or a literal value, such as strings, numbers, etc. Each edge is labeled with a predicate p. Since an edge is represented by a tuple $<s,p,o>$, they are called triples. A URI that takes the subject place in one triple can be the object in another. Note, since ontology instances are represented by URIs assigned in a decentralized fashion, syntactically distinct URIs may actually represent the same entity. For example, a researcher can have multiple distinct URI identifiers in DBLP, ACM and CiteSeer but such URIs actually represent the same real world person; thus they are coreferent. In the Semantic Web, coreferent instances are linked to each other via the ***owl:sameAs*** predicate.

One well-known effort related to entity coreference is Linked Data [1] that enables people to publish their data with links to existing datasets. According to the latest statistics of the Linked Open Data (LOD) cloud[1], there are currently 295 datasets from various domains with more than 31 billion RDF triples and about 500 million links across different datasets. A traditional approach to handling this entity coreference problem is to exhaustively compare every pair of instances in a dataset [2]–[4], which could be extremely expensive for large-scale datasets. Therefore, there is the need to develop scalable entity coreference algorithms to be able to efficiently process large-scale datasets. Furthermore, Halpin et al. [5] reported that only 50% ($\pm$21%) of the owl:sameAs links from the LOD Cloud are correct. This troubling fact emphasizes that although scalability is important, we must also achieve satisfying precision and recall.

In this paper, we propose a pruning-based algorithm for reducing the complexity of entity coreference within a dataset itself. Here, a "dataset" may contain comparable instances from multiple heterogeneous sources. For example, different datasets may use different terms to define the "Researcher" class and ontology instances of all such classes are comparable. First, we design an on-the-fly candidate selection technique to reduce the number of instance pairs to be computed. During the entity coreference process, each instance is compared against other instances; and we hypothesize that two coreferent instances should have similar matching histories, i.e., they should be similar to a sufficiently common set of other instances. We further propose a sigmoid function based thresholding method to automatically adjust the threshold on such history similarity in order to gain a good balance between runtime and F1-score. To speed up the computation for a single pair of instances, we evaluate the potential contribution of their context and only consider the context that is likely to make a significant contribution to their final similarity.

[1]http://www4.wiwiss.fu-berlin.de/lodcloud/state/

We evaluate our system on three Semantic Web instance categories. By comparing to alternatives that only adopt a subset of our proposed techniques, we verify the effectiveness of each individual component. Through the comparison to 9 state-of-the-art systems, we show that our algorithm runs 18 to 24 times faster with comparably good precision and recall. A scalability test on up to 1 million instances shows that the runtime differences become even more substantial as we increase the size of the datasets.

We organize the rest of the paper as follows. Section II provides some background information. We then formally present our pruning-based algorithm in Section III and discuss related work in Section IV. We present our evaluations in Section V and conclude in Section VI.

II. Background

We first introduce a basic entity coreference algorithm *EPWNG* [4], **E**xhaustive **P**airwise Entity Coreference based on **W**eighted **N**eighborhood **G**raph. It computes the similarity between every instance pair in a dataset based on a set of paths, i.e., the context, as shown in Figure 1. A

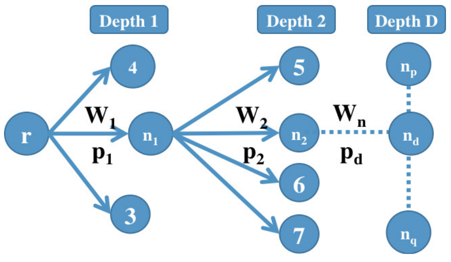


Figure 1. Weighted Neighborhood Graph (G)

path is defined as follows: $path = (r, p_1, n_1, ..., p_d, n_d)$, where r is an ontology instance; n_i and p_i (i>0) are any expanded RDF node and predicate in the path. $N(G, r)$ denotes the context (a set of paths that start from r and end on another RDF node) for r in RDF graph G; $End(path)$ gives the last node in a path. Each node n_i has a weight W_i computed based on the discriminability of its associated predicate p_i; path weight is the multiplication of all its node weights. When collecting context, we adopt breadth-first search and eliminate cycles [4].

Algorithm 1 EPWNG(N_a, N_b), N_a and N_b are the context for instances a and b; returns a and b's similarity

```
1. $score \leftarrow 0, weight \leftarrow 0$
2. for all $paths\ m \in N_a$ do
3.    $n \leftarrow Compare(m, N_b)$;
4.    if $n \neq null$ then
5.       $ps \leftarrow Sim(End(m), End(n))$
6.       $pw \leftarrow (W_m + W_n)/2$
7.       $score \leftarrow score + ps * pw$
8.       $weight \leftarrow weight + pw$
9. return $\frac{score}{weight}$
```

EPWNG (Algorithm 1) compares the comparable paths in the context of two instances a and b. Two paths are comparable if their predicates at corresponding positions are comparable (determined by $PtCmp$), i.e., having the same semantics. For example, predicate *CiteSeer:name* is comparable to *DBLP:name*. Here, we created these

Algorithm 2 Compare(m, N_b), m is a path from N_a; N_b is instance b's context; returns the path of b that is comparable and has the highest similarity to m

```
1. if $\neg\exists path\ n \in N_b,\ PtCmp(m, n)$ then
2.    return $null$
3. if $End(m)\ is\ literal$ then
4.    return $\arg\max_{n \in N_b, PtCmp(m,n)} Sim(End(m), End(n))$
5. else if $End(m)\ is\ URI$ then
6.    if $\exists path\ n \in N_b,\ PtCmp(m, n) \wedge End(m) = End(n)$ then
7.       return $\arg\max_{n \in N_b, PtCmp(m,n) \wedge End(m)=End(n)} W_n$
8.    else
9.       return $null$
```

mapping axioms of predicate comparability manually; but we note that ontology alignment [6], a well studied topic in the Semantic Web, can help automatically determine predicate comparability across multiple ontologies, which is out of the scope of this paper. Sim calculates the string similarity between the last nodes of two paths. For each path m of a, we call *Compare()* (see Algorithm 2) to choose a path n of b that is comparable to and has the highest similarity score to m, denoted as ps. The average weight of path m (W_m) and n (W_n) is used as the weight for ps. The process is repeated for every path of a. The weighted average on such (path score, path weight) pairs is computed as the final similarity score for a and b.

There are two key factors that prevent *EPWNG* from scaling to large-scale datasets. On one hand, *EPWNG* compares every pair of instances in a dataset thus making the entire entity coreference process for large datasets prohibitively expensive. So, one critical question is: ***Can we prune instance pairs that are unlikely to be coreferent to reduce the overall complexity?*** On the other hand, for two instances, *EPWNG* compares all pairs of their comparable paths. Assuming context graphs have branching factor n and depth d, the complexity for computing a single instance pair is $O(n^{2d})$, therefore making *EPWNG* very time-consuming for handling large context. So, another interesting question here is: ***Can we only consider the context that could potentially make a significant contribution to the final similarity score between two instances to further speed up the process?***

III. Algorithm

Algorithm 3 presents the pseudo-code of our proposed pruning-based entity coreference algorithm *P-EPWNG*. At line 1, we adopt a simple yet effective filtering method, i.e., we check if two instances share a single token in the literal values of their top k% most disambiguating context, and if the cosine similarity of such instances' top k% context is above a threshold δ. First, two instances cannot be coreferent if they do not share at least one common word in their context; also, the cosine similarity is used to better handle contexts with many tokens. For example, for publication instances, their titles are generally very disambiguating and thus included in the top k% context; however, only sharing a single token does not necessarily indicate that they could be coreferent in many cases.

From line 3 to 10, we propose an on-the-fly candidate selection technique based upon instances' matching histories to effectively prune out instance pairs that are not

Algorithm 3 P-EPWNG($N_a, N_b, H(a), H(b)$), N_a and N_b are the context for instances a and b; $H(a)$ and $H(b)$ are their histories; returns their similarity

```
1. if $Share\_A\_Token\_Cosine(N_a, N_b, k) \le \delta$ then
2.     return 0;
3. if $HSim(H(a), H(b)) \le Thresholding(a)$ then
4.     return 0;
5. else
6.     $sim\_k \leftarrow EPWNG(kdisc(N_a, k), kdisc(N_b, k))$
7.     if $sim\_k \le \theta$ then
8.         return 0;
9.     else
10.        $H(b) \leftarrow H(b) \cup \{a\}$
11. $score \leftarrow 0, weight \leftarrow 0, s_{curr} \leftarrow 0$
12. $context\_signif \leftarrow \gamma$
13. for all $paths$ $m \in N_a$ do
14.     $n \leftarrow Compare(m, N_b)$;
15.     if $n \ne null$ then
16.         $ps \leftarrow Sim(End(m), End(n))$
17.         $pw \leftarrow (W_m + W_n)/2$
18.         $s_{old} \leftarrow s_{curr}$
19.         $score \leftarrow score + ps * pw$
20.         $weight \leftarrow weight + pw$
21.         $s_{curr} \leftarrow \frac{score}{weight}$
22.         $context\_signif \leftarrow context\_signif - Eval(s_{old}, s_{curr})$
23.         if $context\_signif = 0$ then
24.             return $s_{curr}$
25. return $\frac{score}{weight}$
```

likely to be coreferent. Consider that during the entity coreference process, an instance is compared to many other instances; the results of these prior comparisons could be useful in determining whether two instances might be coreferent. At any point in time, each instance should have a set of other instances that it is somewhat similar to. We define a function $H(a)$ to denote the set of similar instances of instance a, i.e., the matching history. We hypothesize that two coreferent instances should share a sufficient amount of common instances in their histories. Therefore, the general idea behind our on-the-fly candidate selection technique is that before we actually compute the similarity for instances a and b, we adopt a lightweight method to examine if their histories are similar enough, i.e., if $HSim(H(a), H(b))$ is above some threshold (line 3). Instead of utilizing a fixed threshold on $HSim$, we propose a sigmoid function based thresholding method that gradually increases the threshold at runtime until an upper bound is reached, since there is very little history to compare at the beginning. $HSim$ and the thresholding method are further presented in Section III-A.

From line 6 to 10, after an instance pair passes the history check, we should then apply a more expensive method to measure their similarity, which can also be used to determine if we should add one instance to the history of another. In our algorithm, at line 6, we make this decision by computing two instances' similarity by using the *EPWNG* algorithm only with their top k% most disambiguating context (as returned by $kdisc(N_a, k)$). Passing the history check does not necessarily mean the two instances are coreferent and thus it might not make sense to use full context.

Furthermore, we propose a context pruning technique inspired by the idea of quiescence search in game playing. The basic idea is that more time should be spent refining heuristic evaluations when there appear to be significant fluctuations in the value. In this paper, we define an evaluation function to estimate if the remaining context would still make a significant contribution to the final similarity score of two instances. At line 12, we initialize $context\_signif$ (context significance level) with a positive integer γ that can be interpreted as the maximum times that we allow a new path to provide only a small contribution to the final similarity score. Then, at line 22, an evaluation function $Eval$ is used to check the difference between s_{curr} and s_{old}, i.e., if considering one more path still makes a significant change to the similarity score between two instances. We will prune the rest of the context when $context\_signif$ reduces to 0. $Eval$ is further discussed in Section III-B.

A. History-based on-the-fly Candidate Selection

A Modified Jaccard Similarity Measure. As shown in Algorithm 3, we compute the similarity ($HSim$, line 3) between the matching histories of two instances. In this paper, we adopt a modified version of the well-utilized Jaccard similarity measure as defined in Equation 1:

$$HSim(H(a), H(b)) = \begin{cases} +\infty, & H(a)=\emptyset \text{ or } H(b)=\emptyset \quad (1a) \\ \frac{|H(a)\cap H(b)|}{|H(a)\cup H(b)|}, & otherwise \quad (1b) \end{cases}$$

where $H(a)$ represents the matching history of instance a, i.e., a set of other instances that a is similar to. Here, $HSim$ is $+\infty$ when either instance has an empty history, though the traditional Jaccard similarity measure will give a 0 in this case. The intuition is that the history we use for pruning is partial history in the sense that it does not contain a complete set of other instances that the given instance is similar to. Imagine we have a list of instances as shown below:

$$I = < i_1,\ i_2,\ i_3,\ ...,\ i_m,\ i_n,\ ...\ ,\ i_{last} >$$

To detect coreference relationships, we compare each instance with every other instance in the order they are placed, i.e., compare i_1 *to* i_2, i_3, ..., i_{last} and then similarly compare i_2 to all other instances after it. Therefore, when we attempt to compare i_m and i_n, each of them has only been compared with instances that are placed to the left of them and we still do not know their similarities to instances in $\{i_k\}$ ($k>n$). Due to such incomplete knowledge, an instance pair should not be pruned when seeing an empty history.

Sorting Instances. At Line 3 of Algorithm 3, we prune a pair of instances if their history similarity is below a threshold. In our algorithm, this threshold is automatically adjusted at runtime. One approach for performing such adjustment is based upon how many groundtruth coreferent pairs have already been covered by the processed instances at any given time. When more coreferent pairs have been covered, we should adopt a higher threshold in order to prune more un-coreferent pairs. This will lead to runtime savings, without the risk of missing too many true matches. However, for this approach to work, we need a sufficiently sized sample of groundtruth data, but obtaining such a sample for large-scale data is impractical.

To avoid labeling, we compute a ***Match Heuristic*** (*MH*) for each instance before executing Algorithm 3. MH is the number of potential matches of each instance in the dataset. We sort the instances by their MH in descending order, and we call Algorithm 3 on all pairs of an instance with a subsequent instance in descending MH order. To compute MH, we treat the context of each instance as a *bag of words* that we call *doc*. For an instance, *doc* is extracted from all literal paths of length one, i.e., literal values from immediate triples. We then compute a cosine similarity between the *doc* of each instance and that of all other instances with Lucene[2], a well-adopted Information Retrieval tool. We first index all the *doc*s as documents with Lucene. Then each document is treated as a query q and issued to Lucene. We count a returned document doc' as a potential match if $cosine(q, doc')$ calculated by Lucene is above a threshold α.

A Sigmoid Function based Thresholding Approach. There could be different ways to adjust the threshold on the similarity of two instances' matching histories. The simplest way is to use a fixed threshold for the entire coreference process; however, this method may cause the system to miss a certain amount of coreferent pairs at the starting stage since we put instances with the most potential matches at the beginning of the instance list. Therefore, a thresholding approach that starts with a low value but then gradually increases the threshold could potentially be a good choice.

After sorting, each instance is associated with a match heuristic. Intuitively, the match heuristic of an instance can be considered as its weighting factor. When processing instance i_m, $\Sigma_{k=1}^{m-1} MH(i_k)$ gives the total weight of the already processed instances (i.e., their total match heuristics), indicating how important they are and thus can be helpful for thresholding.

In this paper, we propose a sigmoid function based thresholding method that utilizes such match heuristics. The original sigmoid function is defined by Equation 2:

$$F(x) = \frac{1}{1+e^{-x}} \tag{2}$$

The sigmoid function provides a continuous output with limit 1 as x approaches infinity, and limit 0 as x approaches negative infinity.

However, the traditional sigmoid function cannot be directly applied to our situation due to the following two aspects. First of all, the traditional function gives a value of 0.5 when x=0; while we require the computed threshold should be 0 when input is 0. When x is small, the entity coreference process is still in its early stage and thus we have very limited knowledge about the matching histories of the instances. Also, since we sort instances based upon the number of their potential matches in the dataset in descending order, we want to adopt a relatively low threshold at the early stage in the entire entity coreference process to reduce the risk of missing too many true matches. Furthermore, the traditional function has an upper bound of 1. However, in our situation, we may not want to set such a high threshold for filtering and it would be useful if we could adjust the upper bound by setting some parameter.

Based upon the discussion above, we propose a two-phased thresholding function as shown in Equation 3:

$$Thresholding(i_m) = \begin{cases} 0, & x \leq \sigma \quad \text{(3a)} \\ K * F(x(i_m) - 6), & x > \sigma \quad \text{(3b)} \end{cases}$$

where σ determines when to start introducing a threshold; K represents the upper bound of the computed threshold. Input x is the ratio between match heuristics of already processed instances and that of all instances as computed in Eq. 4:

$$x(i_m) = \frac{\Sigma_{k=1}^{m-1} MH(i_k)}{\Sigma_{k=1}^{|I|} MH(i_k)} * 10 \tag{4}$$

where $MH(i)$ is the match heuristic for instance i; the factor 10 enables x to have the range [0, 10] to ensure the output (the computed threshold by Eq. 3) has the ability to transition through most of the range. We shift the original sigmoid curve to the right by replacing x in Equation 2 with x-6, thus the computed threshold gets very close to 0 when x=0.

One alternative to Equation 3 is *Bump*, i.e., setting the history threshold to 0 at the beginning and bumping it to the upper bound when $x>\sigma$. We shall compare these different thresholding approaches in our evaluation (Section V-D).

B. Evaluation Function based Context Pruning

Another important question discussed in Section II is how to reduce the complexity of the comparison for a single pair of instances. Although an instance may have a large number of paths in its context, we should only consider those that can actually make significant contribution to the final similarity score of two instances. In this paper, we define an evaluation function in Equation 5 to judge if we should continue with the remaining paths given the similarity between two instances from their already considered paths:

$$Eval(s_{old},\ s_{curr}) = \begin{cases} 1, & s_{curr} - s_{old} \leq \beta \quad \text{(5a)} \\ 0, & otherwise \quad \text{(5b)} \end{cases}$$

where s_{old} is the similarity between two instances by considering the most highly weighted n-1 paths, and s_{curr} is the similarity by considering the top n paths.

Our evaluation function provides a hint of how important the remaining context is. Linking back to Algorithm 3, at line 12, $context\_signif$ (context significance level) is initialized with γ, the maximum times we can tolerate when considering one more path does not significantly increase the similarity of two instances, i.e., s_{curr}-$s_{old} \leq \beta$. When this happens, $Eval$ will return 1 and consequently at line 22, we reduce our tolerance level of such situations by 1. When the context significance level reaches 0, we suspect all remaining context is insignificant, and we will then decide to ignore the rest of the context to save the overall computational cost. We will show later that adopting our evaluation function can save half of the overall runtime (Section V-D).

[2]http://lucene.apache.org/core/3_6_0/api/core/org/apache/lucene/search/Similarity.html

IV. Related Work

EPWNG [4] adopts a *bag-of-paths* approach to detect coreferent ontology instances. The core idea is that different properties may have quite different impact and thus for each property, a specific weight is automatically assigned. Such property weights are then combined with string matching techniques to compute instance pair similarity. Although it outperforms comparison systems on some benchmark datasets, it took about 17 hours to process 10,000 instances with dense RDF graphs. E/U-EPWNG [7] adopt context pruning techniques that are different from the evaluation function presented in this paper. They utilize sampling techniques and a utility function to prune insignificant context to scale entity coreference but do not prune dissimilar instance pairs.

Aswani et al. [2] try to match person ontology instances converted from the British Telecommunications digital library but their system needs to frequently issue search engine queries to retrieve context information. RiMOM [6], [8] is multi-strategy ontology alignment system designed for alignment at both schema and instance level. Some of the strategies include matching labels with string matching techniques and calculating cosine similarity between context of entities. AgreementMaker [9] and SERIMI [10] are another two algorithms that rely on syntactic string similarity for detecting coreferent ontology instances. Silk [11] indexes ontology instances on manually specified property values for scalability purposes and customized rules are then used to detect coreferent pairs.

Hu et al. [12] build a kernel by adopting the formal semantics of the Semantic Web that is then extended iteratively in terms of discriminative property-value pairs in the descriptions of URIs. Algorithms that combine formal semantics of the Semantic Web and string matching techniques also include Zhishi.me [13], LN2R [14], CODI [15] and ASMOV [16].

Instead of computing every pair of instances in a dataset, candidate selection can be adopted to select candidate instance pairs that are likely to be coreferent in order to reduce the overall search space. ASN [17] learns dynamically sized blocks for each record with a manually determined key. BSL [18] adopted supervised learning to learn a blocking scheme, a disjunction of conjunctions of (method, attribute) pairs. Cao et. al. [19] proposed a similar algorithm that utilizes both labeled and unlabeled data for learning the blocking scheme to reduce the needs of training data. PartEnum [20] is a search based algorithm that adopts a two-level partitioning and enumeration based on Hamming distance. BiTrieJoin [21] is a trie-based method to support efficient edit similarity joins with sub-trie pruning. AllPairs [22] is a simple index based algorithm with certain optimization strategies. PPJoin+ [23] adopts a positional filtering principle that exploits the ordering of tokens in a record. Ed-Join [24] employed filtering methods that explore the locations and contents of mismatching n-grams. Similarly, IndexChunk [25] computes asymmetric signatures on character-level n-grams as constraints for selecting candidates. Differently, FastJoin [26] adopts fuzzy matching techniques that consider both token and character level similarity.

V. Evaluation

Our system is implemented in Java and compiled with Java 6. We conduct all experiments on a RedHat machine with a 12-core Intel 2.8GHz processor and 60GB memory.

A. Evaluation Datasets and Metrics

We evaluate our pruning-based entity coreference algorithm *P-EPWNG* on two Semantic Web datasets: RKB [27] and SWAT[3]. For RKB, we use eight subsets: ACM, DBLP, CiteSeer, EPrints, IEEE, LAAS-CNRS, Newcastle and ECS. Our SWAT dataset consists of RDF data parsed from downloaded XML files of CiteSeer and DBLP. Although both datasets describe the academic domain and share some information, they adopt different ontologies and thus quite different predicates are involved. Also, they may have different coverage in terms of both publication date and venue. We compare on three instance categories: RKB Person, RKB Publication and SWAT Person. The groundtruth is *owl:sameAs* statements and can be crawled from RKB and downloaded from the SWAT website. Since the provided groundtruth was automatically derived and thus incomplete and erroneous, we manually verify and augment the groundtruth to improve their quality.

We adopt the standard metrics, including *Precision*: the number of correctly detected pairs divided by the total number of detected pairs given some threshold; *Recall*: the number of correctly detected pairs divided by the number of coreferent pairs according to the groundtruth; and *F1-score* calculated as $2 * \frac{Precision * Recall}{Precision + Recall}$. Furthermore, we compare the runtime of different systems. In all experiments, we collect instance context off-line and do not include this time in the reported runtime of each system.

For all experiments, we use the same parameter settings as shown in Table I. In general, adopting higher k could improve F1-score a little but slows down the process; using lower γ causes the system to have lower F1-score because it prunes significant context; we can also gain a little higher F1-scores as we increase σ. Due to limited space, we do not present results with other parameter settings.

Table I
Parameter Settings.

Parameter	Value	Description
k (Alg. 3)	5%	the percentage of context used for filtering
θ (Alg. 3)	0.3	if to add an instance to the history of another
γ (Alg. 3)	5	the maximum tolerance level for context pruning
δ (Alg. 3)	0.2	the filtering threshold in *Share_A_Token_Cosine*
α (Sec. III-A)	5	sorting: whether an instance is a potential match
K (Eq. 3)	0.2	the upper bound for history-based pruning
σ (Eq. 3)	30%	deciding when to apply history based pruning
β (Eq. 5)	0.1	expected contribution of computing 1 more path

B. Evaluation Against State-of-the-Art Systems

We show our evaluation results in Table II. We compare *P-EPWNG* to state-of-the-art candidate selection systems: *AllPairs* [22], *PPJoin+* [23], *EdJoin* [24], *FastJoin* [26],

[3]http://swat.cse.lehigh.edu/resources/data/

Table II
EVALUATING AGAINST STATE-OF-THE-ART SYSTEMS. $|Pairs|$: CANDIDATE SET SIZE; $Time$: THE RUNTIME FOR CANDIDATE SELECTION; $Precision$, $Recall$ AND $F1\text{-}score$ ARE THE RELEVANT MEASURES FOR THE ACTUAL ENTITY COREFERENCE PHASE; $Total\ Time$: THE RUNTIME FOR THE ENTIRE PROCESS, INCLUDING BOTH CANDIDATE SELECTION AND ENTITY COREFERENCE.

Dataset	System	*Candidate Selection*		*Coreference*			*Total Time* (s)
		$\|Pairs\|$	*Time (s)*	*Precision (%)*	*Recall (%)*	*F1-score (%)*	
RKB Person	P-EPWNG (Eq. 3)	**68,427**	4.46	95.02	89.52	92.18	**7.66**
	Baseline	N/A	N/A	95.47	83.66	89.15	36.69
	EPWNG [4]	N/A	N/A	93.04	**90.93**	91.96	6,296.91
	U-EPWNG [7]	N/A	N/A	94.22	90.45	92.29	215.50
	Ed-Join [24]	207,643	1.31	95.06	90.74	**92.84**	63.31
	AllPairs [22]	454,972	**0.93**	94.26	90.85	92.52	83.76
	PPJoin+ [23]	454,972	1.02	94.26	90.85	92.52	82.96
	FastJoin [26]	443,846	3.92	94.27	90.85	92.52	81.75
	IndexChunk [25]	953,300	2.27	95.03	90.73	92.82	149.70
	BiTrieJoin [21]	969,637	41.92	**95.65**	84.61	89.77	87.66
	PartEnum [20]	969,637	80.32	**95.65**	84.61	89.77	126.43
SWAT Person	P-EPWNG (Eq. 3)	**43,333**	4.50	99.49	90.88	**94.99**	**8.61**
	Baseline	N/A	N/A	99.37	**90.93**	94.96	31.7
	EPWNG [4]	N/A	N/A	99.45	**90.93**	**94.99**	16968.00
	U-EPWNG [7]	N/A	N/A	99.45	**90.93**	**94.99**	275.17
	Ed-Join [24]	226,330	1.56	99.48	90.81	94.94	102.77
	AllPairs [22]	381,128	**0.79**	99.45	**90.93**	**94.99**	108.34
	PPJoin+ [23]	381,128	0.86	99.45	**90.93**	**94.99**	106.72
	FastJoin [26]	360,481	3.20	99.45	**90.93**	**94.99**	103.72
	IndexChunk [25]	562,114	1.67	99.48	90.84	94.96	210.96
	BiTrieJoin [21]	472,875	22.13	**99.61**	90.29	94.72	124.02
	PartEnum [20]	473,018	70.47	**99.61**	90.29	94.72	123.10
RKB Publication	P-EPWNG (Eq. 3)	**107,471**	13.18	99.66	98.10	98.87	**23.06**
	Baseline	N/A	N/A	98.99	97.28	98.13	72.1
	EPWNG [4]	N/A	N/A	99.78	**99.37**	**99.58**	63200.71
	U-EPWNG [7]	N/A	N/A	**99.82**	99.30	99.56	828.56
	Ed-Join [24]	1,298,525	131.85	99.45	98.36	98.90	1330.20
	AllPairs [22]	581,005	1.41	99.47	99.06	99.27	340.14
	PPJoin+ [23]	581,005	**1.39**	99.47	99.06	99.27	342.21
	FastJoin [26]	583,839	75.78	99.47	99.07	99.27	430.61
	IndexChunk [25]	1,594,764	156.26	99.44	98.05	98.74	1242.85
	BiTrieJoin [21]	N/A	N/A	N/A	N/A	N/A	N/A
	PartEnum [20]	18,776,030	3091.67	98.78	82.27	89.57	47019.50

IndexChunk [25], *BiTrieJoin* [21] and *PartEnum* [20]. Although *P-EPWNG* performs both candidate selection and entity coreference, traditional candidate selection systems only do the first step. Thus, to compare to state-of-the-art systems, we first run those systems on the same input to select candidate instance pairs; and then we apply the *EPWNG* algorithm (the algorithm that does not have our proposed pruning techniques) to the selected pairs. We directly run our proposed algorithm *P-EPWNG* on the same input to get the final entity coreference results.

We also compare to *EPWNG* [4] that performs a brute-force pairwise comparison on all instance pairs without any pruning and to *U-EPWNG* [7] that prunes insignificant context based upon sampling techniques and a utility function without doing candidate selection. A baseline is also tested against where we only compare name (person) and title (publication) for every pair of instances.

We randomly select 100K instances for each instance category. We split each 100K instances into 10 non-overlapping and equal-sized folds, apply all algorithms on each fold and report their average. For coreference results, we report a system's best F1-score from threshold 0.1-0.9.

Baseline. The baseline is not as good as *P-EPWNG* on F1-score and runtime. It has a much lower recall than others on RKB Person. On RKB publication, the baseline has worse precision than *P-EPWNG* since candidate selection helps to filter out some potential false positives. Although we only observed minor difference on F1-score for SWAT Person, the baseline needed significantly more time. For both RKB datasets, both *P-EPWNG* and the baseline have worse recall than *EPWNG*, showing the need to explore context beyond just name and title and thus the need for appropriate context pruning techniques to balance runtime and F1-score. Note that the baseline requires human input on what to compare; also, in the absence of discriminative labels (such as name and title), it may not be able to achieve satisfying results.

EPWNG and U-EPWNG. Overall, compared to our previous entity coreference algorithms *EPWNG* and *U-EPWNG*, *P-EPWNG* achieves substantial runtime savings with slightly worse recall and F1-score. The coreference process was sped up by 2-3 orders of magnitude compared to *EPWNG* and by a factor of 28-36 compared to *U-EPWNG*. There is no surprise that *EPWNG* always achieves the highest recall since no pruning was adopted. Also, *P-EPWNG* achieves a little better precision on both person datasets because pruning dissimilar instance pairs can help to reduce the chance of having false positives.

The State-of-the-Art. Generally speaking, when compared to state-of-the-art candidate selection systems, *P-EPWNG* selects the fewest candidate pairs and thus is able to run the fastest for the entire process, including both candidate selection and coreference. However, since *P-EPWNG* is the most aggressive in pruning instance pairs, its recall and F1-score for the final coreference results are generally slightly worse than that of most of those systems.

On RKB Person, *P-EPWNG* selects 3 (*EdJoin*) to 13 (*IndexChunk*, *BiTrieJoin* and *PartEnum*) times fewer pairs than other systems, which leads to a speedup factor of 8 (*EdJoin*) to 19 (*IndexChunk*) on runtime but also results in 0.66% lower F1-score than the best (achieved by *EdJoin*). Although *P-EPWNG* needs more time for candidate selection, the fact that it selects a lot fewer candidates enables it to achieve substantial runtime savings for the overall process.

We observe similar results for SWAT Person. *P-EPWNG* selects about 5 (*EdJoin*) to 13 (*IndexChunk*) times fewer candidates than other systems; thus the entire process runs 11 (*EdJoin*) to 24 (*IndexChunk*) times faster. Furthermore, *P-EPWNG* is also able to achieve the highest F1-score as *AllPairs*, *PPJoin+* and *FastJoin*.

On RKB Publication, in addition to runtime savings, *P-EPWNG* also achieves the highest precision among all candidate selection systems. *P-EPWNG* prunes instance pairs most aggressively, which helps to reduce the chance of having too many false positives. Note that on this dataset, *P-EPWNG* spends much more time selecting candidates than needed for the person datasets because our *Share_A_Token_Cosine* filtering is not very effective here. *Share_A_Token_Cosine* prunes an instance pair if they do not even share a single token in their most disambiguating context or the cosine similarity between such partial contexts is below a threshold. For person and publication instances, names and titles are generally included in the partial context used by *Share_A_Token_Cosine*; and it is relatively easier for titles to share a token and have a high similarity than names. So, more publication instance pairs can pass this check and are then processed by other more expensive steps.

C. System Scalability

We apply different systems to 100K to 1M instances to examine their scalability. *AllPairs*, *PPJoin+* and *FastJoin* achieve very similar results, so we only compare to *PPJoin+*. We also compare to *EdJoin* on both person datasets where it is better on both the number of selected pairs and the overall runtime (Table II); we did not compare to *EdJoin* on RKB Publication since it selects twice as many pairs as *PPJoin+* on 10K instances and thus is not expected to have better scalability. There are only 500K instances in SWAT Person; also, *PPJoin+* does not scale to 1 million instances on RKB Person.

In Figures 2a to 2c, *P-EPWNG* demonstrates better scalability than the other systems; and it is about 17, 5 and 11 times faster on RKB Publication, RKB Person and SWAT Person respectively on the highest compared scale. The other algorithms demonstrate a clear exponential curve; although *P-EPWNG* is exponential, it has a much smaller exponent. Considering applying these systems to even larger datasets, the runtime differences could become even more substantial.

D. Feature Evaluation

We examine the effectiveness of our proposed pruning techniques by comparing to their alternatives. First, we compare different ways to adjust the threshold on instances' matching history similarity (Section III-A): *Fixed* (using a fixed threshold), *Bump* (starting with 0 and then bumping to a fixed value), *Sigmoid* (σ=0 in Eq. 3) and *M-Sigmoid* (σ=30% in Eq. 3). We also consider removing our context pruning (*M-Sigmoid\Eval*) to examine its impact.

Table III
FEATURE EVALUATION. P: PRECISION, R: RECALL AND F: THE F1-SCORE FOR P AND R; T: THE RUNTIME FOR THE ENTIRE PROCESS.

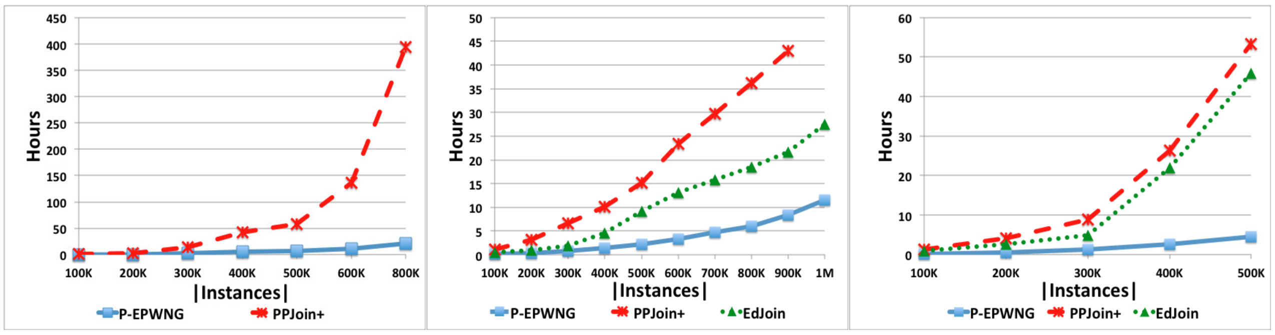
Dataset	System	*P* (%)	*R* (%)	*F* (%)	*T* (s)
RKB Person	M-Sigmoid	95.02	**89.52**	**92.18**	7.66
	Sigmoid	95.05	89.22	92.04	7.56
	Bump	95.14	88.76	91.83	7.51
	Fixed	95.18	88.57	91.75	**7.26**
	M-Sigmoid\Eval	**95.23**	89.28	92.15	14.48
RKB Publication	M-Sigmoid	99.66	**98.10**	**98.87**	23.06
	Sigmoid	99.66	98.03	98.84	22.70
	Bump	99.66	97.91	98.78	22.66
	Fixed	**99.66**	97.80	98.72	**20.86**
	M-Sigmoid\Eval	99.49	98.09	98.79	51.00
SWAT Person	M-Sigmoid	99.49	**90.88**	**94.99**	8.61
	Sigmoid	99.49	**90.88**	**94.99**	8.58
	Bump	99.49	**90.88**	**94.99**	8.32
	Fixed	**99.51**	90.59	94.83	**8.27**
	M-Sigmoid\Eval	99.49	**90.88**	**94.99**	19.78

Table III shows that *M-Sigmoid* has better recall than its thresholding alternatives; with similar precision, it achieves the best F1-scores. *Fixed* runs the fastest by using the upper bound threshold along the whole process. A two-tailed t-test on the F1-scores shows: for RKB Person, the differences between *M-Sigmoid* and *Fixed/Bump*, *Sigmoid* are statistically significant with P values of 0.0001 and 0.0004; for RKB Publication, the differences between *M-Sigmoid* and *Fixed/Bump*, *Sigmoid* are significant with P values of 0.0001 and 0.0101. Furthermore, *M-Sigmoid* is about 1.9 to 2.2 times faster than *M-Sigmoid\Eval*, showing the effectiveness of our evaluation function.

VI. CONCLUSION

In this paper, we propose *P-EPWNG*, a pruning-based algorithm to scalably detect coreference relationships in the Semantic Web. To reduce the impact of number of instances on runtime, an on-the-fly candidate selection technique is proposed for filtering un-coreferent instance pairs if their matching histories are not sufficiently similar. A sigmoid function based thresholding method is also proposed to balance F1-score and runtime. To speed up the computation for a single pair of instances, we further propose an evaluation function to prune insignificant context to their final similarity score. We compare *P-EPWNG* to 9 state-of-the-art candidate selection and entity coreference algorithms and show that our system runs 18 to 24 times faster while only making a small sacrifice in F1-scores (0.71% at maximum) for the final coreference results. Also, a scalability test on up to 1M instances shows runtime savings of 16-370 hours, depending on the dataset. Finally, a feature evaluation verifies the effectiveness of each proposed pruning technique compared to alternatives. For future work, we will apply *P-EPWNG* to datasets from the Ontology Alignment Evaluation Initiative[4] and also evaluate if our pruning techniques can help to scale other entity coreference algorithms [8], [15].

[4]http://oaei.ontologymatching.org



(a) RKB Publication (b) RKB Person (c) SWAT Person

Figure 2. System Scalability

Acknowledgment

This project was partially sponsored by the U.S. Army Research Office (W911NF-11-C-0215). The content of the information does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred.

References

[1] C. Bizer, T. Heath, and T. Berners-Lee, "Linked data - the story so far," *Int. J. Semantic Web Inf. Syst.*, vol. 5, no. 3, pp. 1–22, 2009.

[2] N. Aswani, K. Bontcheva, and H. Cunningham, "Mining information for instance unification," in *International Semantic Web Conference (ISWC)*, 2006, pp. 329–342.

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

[4] D. Song and J. Heflin, "Domain-independent entity coreference for linking ontology instances," *ACM Journal of Data and Information Quality (ACM JDIQ)*, 2012.

[5] H. Halpin, P. J. Hayes, J. P. McCusker, D. L. McGuinness, and H. S. Thompson, "When owl: sameAs isn't the same: An analysis of identity in linked data," in *9th International Semantic Web Conference (ISWC)*, 2010, pp. 305–320.

[6] J. Li, J. Tang, Y. Li, and Q. Luo, "RiMOM: A dynamic multistrategy ontology alignment framework," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 8, pp. 1218–1232, 2009.

[7] D. Song and J. Heflin, "A pruning based approach for scalable entity coreference," in *Proceedings of the Twenty-Fourth International Florida Artificial Intelligence Research Society Conference (FLAIRS)*, 2012, pp. 98–103.

[8] Z. Wang, X. Zhang, L. Hou, Y. Zhao, J. Li, Y. Qi, and J. Tang, "RiMOM results for OAEI 2010," in *Proceedings of the 5th International Workshop on Ontology Matching*, 2010.

[9] I. F. Cruz, F. P. Antonelli, and C. Stroe, "AgreementMaker: Efficient matching for large real-world schemas and ontologies," *PVLDB*, vol. 2, no. 2, pp. 1586–1589, 2009.

[10] S. Araújo, A. P. de Vries, and D. Schwabe, "SERIMI results for OAEI 2011," in *Proceedings of the 6th International Workshop on Ontology Matching (OM)*, 2011.

[11] J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov, "Discovering and maintaining links on the web of data," in *8th International Semantic Web Conference (ISWC)*, 2009, pp. 650–665.

[12] W. Hu, J. Chen, and Y. Qu, "A self-training approach for resolving object coreference on the semantic web," in *Proceedings of the 20th International Conference on World Wide Web (WWW)*, 2011, pp. 87–96.

[13] X. Niu, X. Sun, H. Wang, S. Rong, G. Qi, and Y. Yu, "Zhishi.me - weaving Chinese linking open data," in *International Semantic Web Conference*, 2011, pp. 205–220.

[14] F. Saïs, N. Pernelle, and M.-C. Rousset, "Combining a logical and a numerical method for data reconciliation," *Journal on Data Semantics XII*, vol. 12, pp. 66–94, 2009.

[15] J. Noessner, M. Niepert, C. Meilicke, and H. Stuckenschmidt, "Leveraging terminological structure for object reconciliation," in *7th Extended Semantic Web Conference (ESWC)*, 2010, pp. 334–348.

[16] Y. R. Jean-Mary, E. P. Shironoshita, and M. R. Kabuka, "Ontology matching with semantic verification," *Journal of Web Semantics*, vol. 7, no. 3, pp. 235–251, 2009.

[17] S. Yan, D. Lee, M.-Y. Kan, and C. L. Giles, "Adaptive sorted neighborhood methods for efficient record linkage," in *ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, 2007, pp. 185–194.

[18] M. Michelson and C. A. Knoblock, "Creating relational data from unstructured and ungrammatical data sources," *J. Artif. Intell. Res.*, vol. 31, pp. 543–590, 2008.

[19] Y. Cao, Z. Chen, J. Zhu, P. Yue, C.-Y. Lin, and Y. Yu, "Leveraging unlabeled data to scale blocking for record linkage," in *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, 2011, pp. 2211–2217.

[20] A. Arasu, V. Ganti, and R. Kaushik, "Efficient exact set-similarity joins," in *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB)*, 2006, pp. 918–929.

[21] J. Wang, G. Li, and J. Feng, "Trie-join: Efficient trie-based string similarity joins with edit-distance constraints," *PVLDB*, vol. 3, no. 1, pp. 1219–1230, 2010.

[22] R. J. Bayardo, Y. Ma, and R. Srikant, "Scaling up all pairs similarity search," in *Proceedings of the 16th International Conference on World Wide Web*, 2007, pp. 131–140.

[23] C. Xiao, W. Wang, X. Lin, J. X. Yu, and G. Wang, "Efficient similarity joins for near-duplicate detection," *ACM Trans. Database Syst.*, vol. 36, no. 3, p. 15, 2011.

[24] C. Xiao, W. Wang, and X. Lin, "Ed-join: an efficient algorithm for similarity joins with edit distance constraints," *Proc. VLDB Endow.*, vol. 1, no. 1, pp. 933–944, 2008.

[25] J. Qin, W. Wang, Y. Lu, C. Xiao, and X. Lin, "Efficient exact edit similarity query processing with the asymmetric signature scheme," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2011, pp. 1033–1044.

[26] J. Wang, G. Li, and J. Feng, "Fast-join: An efficient method for fuzzy token matching based string similarity join," in *Proceedings of the 27th International Conference on Data Engineering (ICDE)*, 2011, pp. 458–469.

[27] H. Glaser, I. Millard, and A. Jaffri, "RKBExplorer.com: A knowledge driven infrastructure for linked data providers," in *The 5th European Semantic Web Conference (ESWC)*, 2008, pp. 797–801.