A Pruning Based Approach for Scalable Entity Coreference

Dezhao Song and **Jeff Heflin**
Department of Computer Science and Engineering
Lehigh University
19 Memorial Drive West
Bethlehem, PA 18015

Abstract

Entity coreference is the process to decide which identifiers (e.g., person names, locations, ontology instances, etc.) refer to the same real world entity. In the Semantic Web, entity coreference can be used to detect equivalence relationships between heterogeneous Semantic Web datasets to explicitly link coreferent ontology instances via the ***owl:sameAs*** property. Due to the large scale of Semantic Web data today, we propose two pruning techniques for scalably detecting *owl:sameAs* links between ontology instances by comparing the similarity of their context graphs. First, a sampling based technique is designed to estimate the potential contribution of each RDF node in the context graph and prune insignificant context. Furthermore, a utility function is defined to reduce the cost of performing such estimations. We evaluate our pruning techniques on three Semantic Web instance categories. We show that the pruning techniques enable the entity coreference system to run 10 to 35 times faster than without them while still maintaining comparably good F1-scores.

1 Introduction

The purpose of entity coreference is to decide which identifiers (e.g., person names, publications, geographical locations, etc.) refer to the same real world entity. In the Semantic Web, entity coreference can be used to detect equivalent ontology instances in order to interlink heterogeneous Semantic Web datasets. Here, 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 relationships (class hierarchies and predicates) between these terms. Resource Description Framework (RDF) is a graph based data model for describing resources and their relationships. 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, such as a Universal Resource Identifier (URI) and the object can either be an identifier or a literal value, such as strings, numbers, 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 single person can be represented by different URIs in DBLP and CiteSeer; thus such URIs are coreferent. In the Semantic Web, coreferent instances are linked via the ***owl:sameAs*** predicate and such coreference information can be utilized to facilitate other 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 (Bizer, Heath, and Berners-Lee 2009) is one of the leading efforts that allow people to publish their data with links to other datasets. According to the latest statistics[1], there are currently 207 datasets from various domains in the Linked Open Data (LOD) Cloud with more than 28 billion triples and about 395 million links across different datasets. Therefore, there is the need to develop scalable entity coreference algorithms to be able to handle large scale datasets. On the other hand, as reported by Halpin et al. (2010), only 50% ($\pm$ 21%) of the *owl:sameAs* links from the LOD Cloud are correct. Therefore, while being able to scale to large datasets, an algorithm should still achieve decent precision and recall.

In this paper, we present two novel pruning techniques to build scalable entity coreference system on the Semantic Web. In our algorithm, each instance is associated with a neighborhood graph collected from the entire RDF graph as its context (a set of weighted paths that start from this instance and end on another node in the RDF graph). First, a sampling based pruning technique is developed to estimate the potential contribution of each path in the context in order to prune the context that would not make significant contribution to the final coreference similarity measure. Second, a utility function is defined to measure the cost of performing such estimations in order to further reduce the overall complexity. Compared to an algorithm that considers all paths in the context, our proposed pruning techniques speed up the system by a factor of 10 to 35 while still maintaining comparably good F1-scores on three instance categories.

We organize the rest of the paper as follows. Section 2 discusses related work. In Section 3, we describe our previous

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

entity coreference algorithm based upon which we formally propose our pruning techniques in Section 4. We show our evaluation results in Section 5 and conclude in Section 6.

2 Related Work

The vector space model has been well adopted for entity coreference in free text (Bagga and Baldwin 1998; Gooi and Allan 2004). Han et al. (2004) deploy the Naive Bayes classifier and SVM to disambiguate author names in citations. However, they only evaluated on a few hundred instances. The algorithm proposed by Bhattacharya and Getoor (2007) achieves up to 6% higher F1-score than comparison systems on real world datasets. Their system processed 58,515 person instances in the arXiv dataset in 5 minutes while each instance here only has name information.

In the Semantic Web, Hassel, et al. (2006) propose an algorithm to match ontology instances created from DBLP to DBWorld[2] mentions. They selectively pick some triples (e.g., name and affiliation) of an instance to match the context in free text and achieve good results. However, only about 800 entities were involved in their coreference task. Aswani et al. (2006) try to match person ontology instances converted from the British Telecommunications digital library, containing 4,429 publications and 9,065 author names. They issue queries to search engines to find context information and achieve good results; however, the system may not scale to large datasets since it needs to frequently interact with the web to retrieve context information. In previous work, we adopt a *bag-of-paths* approach to detect coreferent ontology instances (2010). The core idea is that different properties may have quite different impact and thus for each property, a specific weight is automatically assigned. Combining such property weights with string matching techniques, the similarity between two instances is computed. Although it outperforms comparison systems on some benchmark datasets, it took about 4 hours to process 10K instances in datasets with dense RDF graphs.

Hu et al. (2011) adopt a two-step approach for detecting coreferent instances. For a URI, they firstly establish a kernel that consists of semantically coreferent URIs based on *owl:sameAs*, (inverse) functional properties and (max-) cardinalities; then they extend such kernel iteratively in terms of discriminative property-value pairs in the descriptions of URIs. The system was tested on a large dataset where an instance has 7.81 triples on average and it needs about 8.6 seconds to detect the *owl:sameAs* links for a single instance. Similar algorithms also include LN2R (Saïs, Pernelle, and Rousset 2009), CODI (Noessner et al. 2010) and ASMOV (Jean-Mary, Shironoshita, and Kabuka 2009).

3 Preliminaries

We first review our previous context and bag-of-paths based entity coreference algorithm (2010) based upon which our pruning techniques are proposed and applied. We refer to it as $Naive$ here. $Naive$ detects coreferent instances by comparing their neighborhood graphs (the context), a set of collected paths, as shown in Figure 1. A path is defined as follows: $path = (r, p_1, n_1, ..., p_n, n_n)$, 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 predicate is assigned a weight automatically (W) based on its discriminability and the weight of a path is the multiplication of all its predicate weights (2010).

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

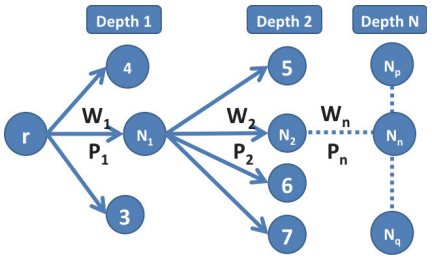

Figure 1: Weighted Neighborhood Graph (G)

Algorithm 1 Naive(N_a, N_b), N_a and N_b are the context for instances a and b respectively; returns the similarity between a and b

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

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 to and has the highest similarity to m

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

$Naive$ (Algorithm 1) adopts the *bag-of-paths* approach to compare the comparable paths in the context of two ontology instances. $PtCmp$ determines if two paths are comparable; Sim calculates the string similarity between the last nodes of two paths. For each path (m) of instance a, the algorithm compares its last node to that of every comparable path of instance b and chooses the highest similarity score, denoted as ps. To assign a weight for ps, the average of the weight (W_m) of m and the weight (W_n) of path n of instance b that has the highest similarity to m is used. The process is repeated for every path of a. A weighted average on such (path score, path weight) pairs is computed to be the final similarity score between the two instances.

The $Naive$ algorithm can be simplified to be Equation 1:

$$Sim(a,b) = \frac{\Sigma_{i \in paths}(pw_i * ps_i)}{\Sigma_{i \in paths} pw_i} \quad (1)$$

where $paths$ denotes the paths of instance a; i is one of such paths; ps_i and pw_i are the maximum path similarity of path i to its comparable paths from instance b and the corresponding path weight respectively. Assuming context graphs have branching factor n and depth d, then the runtime complexity of $Naive$ for a single pair of instances is $O(n^{2d})$. The key factor here is the number of paths contained in the context. $Naive$ can be very time-consuming for large context. Therefore, one important question here is: ***Can we only consider paths that could potentially make a significant contribution to the final similarity score between two instances to reduce the overall computational cost?***

4 Algorithm

In this section, we propose two pruning techniques to reduce the overall complexity of the $Naive$ algorithm. Although an instance may have a large number of paths in its context, only those that could potentially make a significant contribution to the final similarity measure should be considered. Based upon this idea, Equation 1 is changed to Equation 2:

$$Sim(a,b) = \frac{\Sigma_{i \in paths'}(pw_i * ps_i) + \Sigma_{j \in paths''}(pw_j * est_j)}{\Sigma_{i \in paths'} pw_i + \Sigma_{j \in paths''} pw_j} \quad (2)$$

where i is one of the paths of instance a that have already been considered by the algorithm; j is one of the paths that have not been covered; est_j and pw_j are the estimated similarity (the potential contribution) of path j to its comparable paths from instance b and its path weight respectively; the combination of $paths'$ and $paths''$ is the entire context of a.

The intuition is that an entity coreference algorithm could safely ignore the rest of the context of an instance when it reaches a boundary. This boundary is a place where the contribution of the remaining context cannot over turn the current decision made based upon the already considered context. In other words, with the estimated path similarity est_j for the remaining paths of instance a, if the similarity between the two instances cannot be greater than a pre-defined threshold, the algorithm should stop at the current path to save computational cost, i.e. it prunes the rest of the context.

Algorithm 3 shows the pseudo code of the modified algorithm. The key modification is at line 3. The algorithm $Continue$ determines if $ComparePruning$ should continue to process the next path in the context by estimating the similarity between a and b based on the potential similarity score of the end node of each remaining path in instance a's context. In the $Continue$ algorithm, $Utility$ denotes a utility function to determine if it is worth performing such an estimation (line 5-14): 1) if we do not want to use the utility function (i.e., $samplingOnly$ is true), we will directly go to line 5 to perform an estimation; 2) if we use the utility function, we calculate the utility of performing an estimation (u). If this utility is less than 0 ($u < 0$), we will return true to process the next path; otherwise, we perform an estimation.

Algorithm 3 ComparePruning($samplingOnly$, N_a, N_b), N_a and N_b are the context collected for instances a and b respectively; $samplingOnly$ indicates if the algorithm will use the utility function; returns the similarity between a and b

```
1.  score ← 0, weight ← 0
2.  for all paths m∈N_a do
3.      if Continue(samplingOnly, score, weight, N_a, Pos(m)) then
4.          if ∃path n∈N_b, PtCmp(m, n) then
5.              n ← Comparison(m, N_b);
6.              if n ≠ null then
7.                  ps ← Sim(End(m), End(n))
8.                  pw ← (W_m + W_n)/2
9.                  score ← score + ps * pw
10.                 weight ← weight + pw
11.     else
12.         return 0
13. return total_score / total_weight
```

We shall present the details of the $Continue$ algorithm in the rest of this section.

Sampling Based End Node Similarity Estimation

Algorithm 4 presents the details of the $Continue$ function adopted in Algorithm 3. As discussed in Section 3, each path has a weight and here we *prioritize* the paths based upon their weight, i.e., paths with higher weight will be considered first. A perfect match on high-weight paths indicates that the algorithm should continue to process the remaining context; while a mismatch on high-weight paths could help the algorithm to stop at appropriate places for non-coreferent instance pairs before wasting more efforts. Here,

Algorithm 4 Continue($samplingOnly$, $score$, $weight$, N_a, $index_m$), $samplingOnly$ indicates if the algorithm will use the utility function; $score$ and $weight$ are the sum of the end node similarity and their corresponding weight of the already considered paths; N_a is the context of instance a; $index_m$ is the index of path m; returns a boolean value

```
1.  if samplingOnly is false then
2.      u ← Utility(index_m, |N_a|)
3.      if u < 0 then
4.          return true
5.  current ← score / weight
6.  for all paths m'∈N_a do
7.      m'.est ← the estimated end node similarity for path m'
8.      if m' has not been considered and m'.est ≥ current then
9.          score ← score + m'.est * m'.weight
10.         weight ← weight + m'.weight
11. if score / weight > θ then
12.     return true
13. else
14.     return false
```

$score$ and $weight$ are the sum of the end node similarity and their corresponding path weight. They represent the similarity ($current$) between two instances based upon the already considered context. When calculating the potential similarity score between the two instances, we only consider the remaining paths whose estimated node similarity ($m'.est$) is no less than $current$ (line 7) since paths whose estimated end node similarity is smaller than $current$ will only lower the final similarity measure.

Our entity coreference algorithm ($CompareePruning$) computes coreference relationships by measuring the similarity between end nodes of two paths. So, one key factor to apply our pruning technique is to appropriately estimate the similarity that the last node of a path could potentially have with that of its comparable paths of another instance (est in Equation 2), i.e., the potential contribution of each path. The higher similarity that a path has, the more contribution it could make to the final score between two instances.

Since our algorithm only checks if two URIs are identical, Equation 3 is used to estimate URI end node similarity for an object value of a set of comparable properties:

$$est(G, P, obj) = \frac{\{t|t =< s, p, obj > \wedge\, t \in G\}}{\{t|t =< s, p, x > \wedge\, t \in G\}}, p \in P \tag{3}$$

where G is an RDF graph; P is a set of comparable object properties; obj is a specific object for any property in P; t is a triple and x represents any arbitrary object value of properties in P. It represents how likely one URI node would meet an identical node in RDF graph G and we calculate it as the estimated similarity for each specific object of property $p \in P$. Similarly, we could compute the estimated similarity for the subject values of all object properties.

To estimate for literal nodes, we randomly select a certain number (ϵ) of literal values of a property, conduct a pairwise comparison among all the selected literals, and finally get the estimated similarity score as shown in Equation 4 (see next page). Here, P is a set of comparable datatype properties; $Subset(G, P)$ randomly selects a certain number of literal values of P, such as o_1 and o_2 whose similarity is computed with the function Sim; γ is a percentage value that controls how many pairwise comparisons should be covered in order to give the estimated node similarity. The intuition here is to find a sufficiently high similarity score as the potential similarity between two literal values of P in order to reduce the chance of missing too many coreferent pairs. In this case, such estimation is calculated with respect to each individual property.

Utility Based Decision Making

As described in Algorithm 3, before we actually process each path in the context, we perform an estimation (line 3, $samplingOnly$ being true) such that if the potential similarity between two instances would be below a threshold, we just stop considering the rest of the context. However, performing estimations has a computational cost in the entity coreference process. Suppose the algorithm stops after considering k paths, then k estimations were actually performed according to the sampling based technique. However, if the algorithm knew that it would stop after considering k paths, it should have only performed one estimation at the kth path. To maximally avoid those unnecessary estimations, we ask: ***Can we perform estimations only when needed?***

Based upon the discussion above, in order to further reduce the overall complexity of the entity coreference process, we define a utility function as shown in Figure 2. Suppose we reach a decision node ***d/n*** (there are n paths in total and the algorithm is now at the dth path), we would then

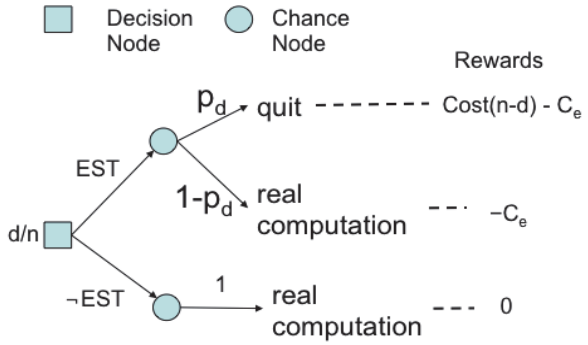

Figure 2: Utility Function

need to decide whether we want to perform an estimation (EST vs. $\neg EST$). The general decision making process is described as follows:

- If we choose not to do estimation ($\neg EST$), then the only choice is to perform a real computation. For a path m of instance a, the algorithm will find all paths comparable to m from the context of instance b and compute the similarity by following line 4 to 10 in Algorithm 3. We will not have any rewards by going this route;
- If we perform an estimation at node d/n (EST), then there could be two different outcomes:
  - The algorithm will quit (with probability p_d) because it estimates that this pair of instances are not similar to a pre-defined level (their estimated similarity score is lower than θ in Algorithm 4). In this case, we have rewards $Cost(n-d)$, where n is the total number of paths of instance a, d is the current path number, $Cost(n-d)$ is the cost of performing real computations for the rest $n-d$ paths, i.e., $Cost(n-d) = (n-d) * C_r$ with C_r being the cost of doing a real computation for a path. Here, we also spent some time (C_e) for an estimation;
  - If the algorithm continues based upon the estimation results, then the rewards will be $-C_e$ because there is no gain but an estimation has been performed.

Summarizing all possibilities, the utilities for actions EST and $\neg EST$ are formally defined in Equations 5 and 6:

$$EU(EST) = p_d * (Cost(n-d) - C_e) + (1-p_d) * (-C_e) \tag{5}$$

$$EU(\neg EST) = 0 \tag{6}$$

where p_d is the probability that the algorithm stops at the dth path. We perform an estimation when the marginal utility given in Equation 7 is a positive value (line 2 of Algorithm 4) otherwise a real computation is executed.

$$\begin{aligned} Utility &= EU(EST) - EU(\neg EST) = \\ & p_d * (n-d) * C_r - p_d * C_e - C_e + p_d * C_e = \\ & p_d * (n-d) * C_r - C_e \end{aligned} \tag{7}$$

To estimate the parameters for each category of instances, we randomly select a small number of instances (α) for each

$$est(G,P) = \arg\min_{score} \frac{|\{(o_1,o_2)|o_1,o_2 \in Subset(G,P) \wedge Sim(o_1,o_2) \leq score\}|}{|\{(o_1,o_2)|o_1,o_2 \in Subset(G,P)\}|} > \gamma \quad (4)$$

category and run the sampling based algorithm on them. Then, we compute the average time for performing an estimation (C_e) and a real computation (C_r) respectively. We adopt Equation 8 to estimate the probability that the algorithm stops at the dth path (p_d).

$$p_d = \frac{|The\ algorithm\ stopped\ at\ the\ dth\ path|}{|The\ algorithm\ passed\ d\ paths|} \quad (8)$$

5 Evaluation

We evaluate our pruning techniques on two RDF datasets: RKB (Glaser, Millard, and Jaffri 2008) and SWAT[3]. The SWAT RDF dataset was parsed from the downloaded XML files of CiteSeer and DBLP. Both datasets describe publications and share some information; but they use different ontologies and their coverage is also different. We compare on 3 instance categories with 100K randomly selected instances for each: RKB Person, RKB Publication and SWAT Person. The groundtruth was provided as *owl:sameAs* statements and can either be crawled from RKB or downloaded from SWAT.

Our system is implemented in Java and we conduct all experiments on a Sun Workstation with an 8-core Intel 2.93GHz processor and 3GB memory. We compare the performance of 3 systems: Naive (Song and Heflin 2010), the sampling based algorithm (Sampling) and the system that combines sampling and the utility function (Utility). We report a system's best F1-Score (for precision and recall) from threshold 0.3-0.9. We split each 100K dataset into 10 non-overlapping and equal-sized subsets, run all algorithms on the same input and report the average. We also test the statistical significance on the results of the 10 subsets from two systems via a two-tailed t-test. On average, there are 6,096, 4,743 and 684 coreferent pairs for each subset of RKB Person, RKB Publication and SWAT Person respectively.

We have the following parameters in our system: γ (Equation 4) is the percentage of pairwise similarities to be covered to estimate literal node similarity; θ (Algorithm 4) determines if an instance pair is still possible to be coreferent; ϵ is the number of literal values selected for literal node similarity estimation; and α is the number of instances to be used for estimating the parameters in our utility function. We set γ, θ, ϵ and α to be 0.95, 0.2, 1,000 and 200 respectively and use the same values for all experiments and it took 110 and 78 seconds to estimate literal end node similarity for the RKB dataset and the SWAT dataset respectively.

Pruning Based Entity Coreference Results

Table 1 shows the evaluation results by applying the three different entity coreference algorithms on the ten subsets of the three 100K subsets. $Speedup$ is the speedup factor on the runtime of different algorithms computed as following: $Speedup = \frac{Runtime\ of\ Naive}{Runtime\ of\ Baseline/Sampling/Utility}$. We also compare to a baseline where we simply choose paths whose weight is higher than a threshold β (0.01, 0.02,..., 0.10). Here, we use low values for β since the calculated path weights are typically very low.

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

Table 1: Entity Coreference Results. P and R represent precision and recall respectively; F1 is the F1-score for P and R; Baseline(β) means only using paths with weight higher than β.

Dataset	System	P	R	$F1$	$Speedup$
RKB Person	Naive	94.04	90.13	92.02	1
	Baseline (0.02)	95.49	87.58	91.33	9.47
	Baseline (0.03)	95.75	87.15	91.22	13.01
	Baseline (0.04)	95.51	86.00	90.48	16.28
	Baseline (0.05)	95.64	84.98	89.97	20.57
	Baseline (0.06)	**95.83**	79.72	87.00	**23.09**
	Sampling	94.02	**90.46**	92.21	10.43
	Utility	94.21	90.39	**92.26**	15.61
RKB Pub	Naive	99.69	99.13	99.41	1
	Baseline (0.01)	99.61	**99.45**	**99.53**	32.39
	Baseline (0.02)	**99.71**	98.95	99.33	56.63
	Baseline (0.03)	99.69	96.83	98.24	59.64
	Baseline (0.04)	99.56	93.25	96.30	61.13
	Baseline (0.05)	99.56	87.42	93.10	**63.26**
	Sampling	99.44	98.81	99.13	22.62
	Utility	99.44	98.97	99.21	29.11
SWAT Peron	Naive	99.29	91.24	**95.07**	1
	Baseline (0.01)	99.27	**91.25**	**95.07**	12.57
	Baseline (0.10)	**99.70**	90.85	95.06	34.57
	Sampling	99.28	91.24	**95.07**	28.72
	Utility	99.28	91.24	**95.07**	**34.62**

With our proposed pruning techniques, both $Sampling$ and $Utility$ run much faster than $Naive$. Particularly, with the utility function, the system $Utility$ was able to achieve even higher speedup factors than $Sampling$. On the other hand, while successfully scaling the $Naive$ algorithm, both $Sampling$ and $Utility$ still maintain comparable F1-scores to that of $Naive$ on all datasets. As shown in Table 1, they both achieve even higher F1-scores than $Naive$ on RKB Person, although their F1-score is only slightly lower than that of $Naive$ on RKB Publication. The improvements on the person datasets come from higher precision since the pruning techniques can help to remove some paths where non-coreferent instances happen to have similar values (e.g., two distinct person instances could have identical date for their publications). A statistical test on the F1-scores shows that the difference between $Naive$ and $Sampling$/$Utility$ on RKB Publication is significant with P values of 0.0001 and 0.0024 respectively.

Compared to the baseline, $Utility$ achieves better or equally good F1-scores on person datasets, though it is not as good as the baseline on RKB Publication. On RKB Person, the baseline (β=0.04-0.06) has higher speedup with significantly lower F1; when β is low, it has higher F1-scores at the cost of runtime. On SWAT Person, the baseline (β=0.10) achieves a comparable speedup factor to $Utility$ with minor

difference in F1-score. Surprisingly, on RKB Publication, our simple baseline (β=0.01,0.02) has better performance than $Utility$ on both F1-score and speedup factor. One possible reason could be that matching publications is relatively easy since the titles can generally provide sufficient information; thus simply cutting off those redundant paths by setting appropriate thresholds could greatly speed up the process while achieving good F1-scores.

Although the baseline outperforms $Utility$ on RKB Publication, the actual users will probably have to tune the system to find out the best β values for different datasets. According to Table 1, 0.03, 0.01 and 0.10 could be the best β values for RKB Person, RKB Publication and SWAT Person respectively. However, if we adopt the same value (e.g., 0.10) in all cases, the baseline could potentially suffer from having low F1-scores on some datasets. Thus, the baseline may not be the best choice in terms of generality.

Scaling to Larger Scale

We run $Naive$, $Sampling$ and $Utility$ on up to 20K randomly selected instances from each of the three instance categories to show the capability of the proposed pruning techniques in scaling the entity coreference process at different scales. We only test on up to 20K instances since $Naive$ does not scale to larger scale due to insufficient memory.

Figure 3 shows the runtime speedup factor of $Sampling$ and $Utility$ compared to $Naive$ on the three instance categories. $Sampling$ alone enables the entity coreference process to run 20 to 30 times faster. When combined with the utility function, the system $Utility$ achieves a significantly higher speedup factor of about 25 to 40 for all three datasets. The actual runtime (seconds) for $Utility$ to process 20K instances is 700, 2,934 and 878 for RKB Person, RKB Publication and SWAT Person respectively.

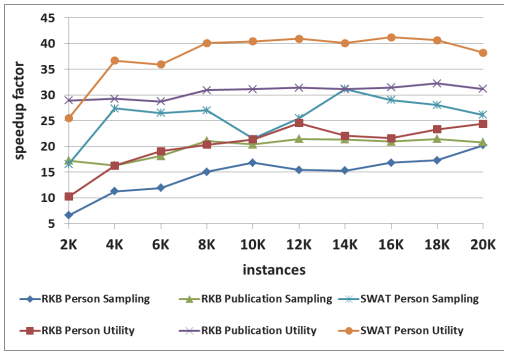

Figure 3: Runtime Speedup Factor

6 Conclusion

In this paper, we propose two sampling and utility function based pruning techniques to build a scalable entity coreference system. With the sampling technique, we estimate the potential contribution of RDF nodes in the context of an instance and prune the context that would not make significant contribution to the final coreference similarity measure. Furthermore, we define a utility function that measures the cost of performing such estimation in order to avoid those unnecessary estimations. We evaluate our proposed pruning techniques on three Semantic Web instance categories. While maintaining comparably good F1-scores, our system runs 15 to 35 times faster than a naive system that considers all RDF nodes in a given context. For future work, it would be interesting to explore how to automatically learn the appropriate parameter values. Also, we will test our system on more domains, such as geographic and life sciences.

7 Acknowledgments

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

Aswani, N.; Bontcheva, K.; and Cunningham, H. 2006. Mining information for instance unification. In *The 5th 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.

Bhattacharya, I., and Getoor, L. 2007. Collective entity resolution in relational data. *TKDD* 1(1).

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.

Glaser, H.; Millard, I.; and Jaffri, A. 2008. Rkbexplorer.com: A knowledge driven infrastructure for linked data providers. In *The 5th European Semantic Web Conference (ESWC)*, 797–801.

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

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 *5th International Semantic Web Conference (ISWC)*, 44–57.

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

Jean-Mary, Y. R.; Shironoshita, E. P.; and Kabuka, M. R. 2009. Ontology matching with semantic verification. *Journal of Web Semantics* 7(3):235–251.

Noessner, J.; Niepert, M.; Meilicke, C.; and Stuckenschmidt, H. 2010. Leveraging terminological structure for object reconciliation. In *7th Extended Semantic Web Conference (ESWC)*, 334–348.

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 Conference on Information and Knowledge Management (CIKM)*, 1821–1824.