# Automatically Generating Data Linkages Using a Domain-Independent Candidate Selection Approach

Dezhao Song and Jeff Heflin

Department of Computer Science and Engineering, Lehigh University,
19 Memorial Drive West, Bethlehem, PA 18015, USA
{des308,heflin}@cse.lehigh.edu

**Abstract.** One challenge for Linked Data is scalably establishing high-quality *owl:sameAs* links between instances (e.g., people, geographical locations, publications, etc.) in different data sources. Traditional approaches to this entity coreference problem do not scale because they exhaustively compare every pair of instances. In this paper, we propose a candidate selection algorithm for pruning the search space for entity coreference. We select candidate instance pairs by computing a character-level similarity on discriminating literal values that are chosen using domain-independent unsupervised learning. We index the instances on the chosen predicates' literal values to efficiently look up similar instances. We evaluate our approach on two RDF and three structured datasets. We show that the traditional metrics don't always accurately reflect the relative benefits of candidate selection, and propose additional metrics. We show that our algorithm frequently outperforms alternatives and is able to process 1 million instances in under one hour on a single Sun Workstation. Furthermore, on the RDF datasets, we show that the entire entity coreference process scales well by applying our technique. Surprisingly, this high recall, low precision filtering mechanism frequently leads to higher F-scores in the overall system.

**Keywords:** Linked Data, Entity Coreference, Scalability, Candidate Selection, Domain-Independence.

## 1 Introduction

One challenge for the Linked Data [4] is to scalably establish high quality *owl:sameAs* links between instances in different data sources. According to the latest statistics[1], there are currently 256 datasets (from various domains, e.g., Media, Geographic, Publications, etc.) in the Linked Open Data (LOD) Cloud with more than 30 billion triples and about 471 million links across different datasets. This large volume of data requires automatic approaches be adopted for detecting *owl:sameAs* links. Prior research to this entity coreference problem[2]

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

[2] Entity Coreference is also referred to as Entity Resolution, Disambiguation, etc.

L. Aroyo et al. (Eds.): ISWC 2011, Part I, LNCS 7031, pp. 649–664, 2011.
© Springer-Verlag Berlin Heidelberg 2011

[1,10,15] has focused on how to precisely and comprehensively detect coreferent instances and good results were achieved. However, one common problem with previous algorithms is that they were only applied to a small number of instances because they exhaustively compare every pair of instances in a given dataset. Therefore, such algorithms are unlikely to be of practical use at the scale of Linked Data. Although Sleeman and Finin [14] adopted a filtering mechanism to select potentially matching pairs, their filter checks every pair of instances by potentially having to consider all associated properties of an instance; this is unlikely to scale for datasets with many properties.

To scale entity coreference systems, one solution would be to efficiently determine if an instance pair could be coreferent by only comparing part of the pair's context, i.e., candidate selection. Other researchers have used the term *blocking* [12] but with two different meanings: finding non-overlapping blocks of instances such that all instances in a block will be compared to each other or simply locating similar instance pairs. This second usage is what we refer to as *candidate selection* in this paper. For an instance, we select other instances that it could be coreferent with, i.e., selecting a candidate set of instance pairs. Several interesting questions then arise. First, manually choosing the information to compare might not work for all domains due to insufficient domain expertise. Also, candidate selection should cover as many true matches as possible and reduce many true negatives. Finally, the candidate selection algorithm itself should scale to very large datasets.

In this paper, we propose a candidate selection algorithm with the properties discussed above. Although our algorithm is designed for RDF data, it generalizes to any structured dataset. Given an RDF graph and the types of instances to do entity coreference on, through unsupervised learning, we learn a set of datatype properties as the candidate selection key that both discriminates and covers the instances well in a domain-independent manner. We then utilize the object values of such predicates for candidate selection. In order to support efficient look-up for similar instances, we index the instances on the learned predicates' object values and adopt a character level n-gram based string similarity measure to select candidate pairs. We evaluate our algorithm on 3 instance categories from 2 RDF datasets and on another 3 well adopted structured datasets for evaluating entity coreference systems. Instead of only using traditional metrics (to be described in Section 2) for evaluating candidate selection results, we propose to apply an actual entity coreference system to the selected candidate pairs to measure the overall runtime and the final F1-score of the coreference results. We show that our proposed algorithm frequently outperforms alternatives in terms of the overall runtime and the F1-score of the coreference results; it also commonly achieved the best or comparably good results on the non-RDF datasets.

We organize the rest of the paper as following. We discuss the related work in Section 2. Section 3 presents the process of learning the predicates for candidate selection and Section 4 describes how to efficiently look up and select candidate instance pairs by comparing the object values of the learned predicates. We evaluate our algorithm in Section 5 and conclude in Section 6.

## 2 Related Work

Several candidate selection algorithms have been proposed. Best Five [18] 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. ASN [21] learns dynamically sized blocks for each record with a manually determined key. The authors claim that changing to different keys didn't affect the results but no data was reported. Marlin [3] 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 datasets, it only reduced the pairs to consider by 55.35%.

BSL [12] 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; 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 [12], compared to using 50% of the groundtruth for training, 4.68% fewer true matches were covered on some dataset by training on only 10% of the groundtruth. In order to reduce the needs of training data, Cao et. al. [5] proposed a similar algorithm that utilizes both labeled and unlabeled data for learning the blocking scheme.

Adaptive Filtering (AF) [9] is unsupervised and is similar to our approach in that it filters record pairs by computing their character level bigram similarity. All-Pairs [2], PP-Join(+) [20] and Ed-Join [19] are all inverted index based approaches. 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 proposed filtering methods that explore the locations and contents of mismatching n-grams. Silk [17] indexes ontology instances on the values of manually specified properties to efficiently retrieve similar instance pairs. Customized rules are then used to detect coreferent pairs.

Compared to Best Five and ASN, our approach automatically learns the candidate selection key for various domains. Unlike Marlin, our system can both effectively reduce candidate set size and achieve good coverage on true matches. Although BSL achieved good results on various domains, its drawbacks are that it requires sufficient training data and is not able to scale to large datasets [13]. Cao et. al. [5] used unlabeled data for learning. However the supervised nature of their method still requires a certain amount of available groundtruth; while our algorithm is totally unsupervised. Similar to AF and Ed-Join, we also exploit using n-grams. However, later we show that our method covers 5.06% more groundtruth than AF on a census dataset and it generally selects one order of magnitude (or even more) fewer pairs than Ed-Join. All-Pairs and PP-Join(+) treat each token in a record as a feature and select features by only considering their frequency in the entire document collection; while we select the information for candidate selection on a predicate-basis and consider both if a predicate discriminates well and if it is used by a sufficient number of instances.

The Ontology Alignment Evaluation Initiative (OAEI) [7] includes an instance matching track that provides several benchmark datasets to evaluate entity coreference systems for detecting equivalent ontology instances; however, some of the datasets are of small scale and thus cannot sufficiently demonstrate the scalability of a candidate selection algorithm. Three metrics have been well adopted for evaluating candidate selection (Eq. 1): Pairwise Completeness ($PC$), Reduction Ratio ($RR$) and F-score ($F_{cs}$) [6,21]. $PC$ and $RR$ evaluate how many true positives are returned by the algorithm and the degree to which it reduces the number of comparisons needed respectively; $F_{cs}$ is their F1-score, giving a comprehensive view of how well a system performs.

$$PC = \frac{|true\ matches\ in\ candidate\ set|}{|true\ matches|}, RR = 1 - \frac{|candidate\ set|}{N * M} \quad (1)$$

where $N$ and $M$ are the sizes of two instance sets that are matched to one another. As we will show in Section 5.3, when applied to large datasets (with tens of thousands of instances), a large change in the size of the candidate set may only be reflected by a small change in $RR$ due to its large denominator.

## 3 Learning the Candidate Selection Key

As discussed, candidate selection is the process of efficiently selecting possibly coreferent instance pairs by only comparing part of their context information. Therefore, the information we will compare needs to be useful in disambiguating the instances. For example, a person instance may have the following triples:

| | | |
|---|---|---|
| person#100 | *has-last-name* | "Henderson" |
| person#100 | *has-first-name* | "James" |
| person#100 | *lives-in* | "United States" |

Intuitively, we might say that last name could disambiguate this instance from others better than first name which is better than the place where he lives in. The reason could be that the last name *Henderson* is less common than the first name *James*; and a lot more people live in the United States than those using *James* as first name. Therefore, for person instances, we might choose the object values of *has-last-name* for candidate selection. However, we need to be able to automatically learn such disambiguating predicate(s) in a domain-independent manner. Furthermore, the object values of a single predicate may not be sufficiently disambiguating to the instances. Take the above example again, it could be more disambiguating if we use both last name and first name.

Algorithm 1 presents the process for learning the candidate selection key, a set of datatype predicates, whose object values are then utilized for candidate selection. Triples with datatype predicates use literal values as objects. The goal is to iteratively discover a predicate set (the candidate selection key) whose values are sufficiently discriminating (discriminability) such that the vast majority of instances in a given dataset use at least one of the learned predicates (coverage). The algorithm starts with an RDF graph G (a set of triples, <i,p,o>)

**Algorithm 1.** Learn_Key($G$, $C$), $G$ is an RDF graph, consisting a set of triples, $C$ is a set of instance types

1. $key\_set \leftarrow$ *a set of datatype properties in* $G$
2. $I_C \leftarrow \{i| < i,$rdf:type$, c > \in G \wedge c \in C\}$
3. $satisfied \leftarrow$ **false**
4. **while not** $satisfied$ **and** $key\_set \neq \emptyset$ **do**
5. **for** $key \in key\_set$ **do**
6. $discriminability \leftarrow dis(key, I_C, G)$
7. **if** $discriminability < \beta$ **then**
8. $key\_set \leftarrow key\_set - key$
9. **else**
10. $coverage \leftarrow cov(key, I_C, G)$
11. $F_L \leftarrow \frac{2*discriminability*coverage}{discriminability+coverage}$
12. $score[key] \leftarrow F_L$
13. **if** $F_L > \alpha$ **then**
14. $satisfied \leftarrow$ **true**
15. **if not** $satisfied$ **then**
16. $dis\_key \leftarrow \arg\max_{key \in key\_set} dis(key, I_C, G)$
17. $key\_set \leftarrow$ *combine* $dis\_key$ *with all other keys*
18. $G \leftarrow update(I_C, key\_set, G)$
19. **return** $\arg\max_{key \in key\_set} score[key]$

and it extracts all the datatype predicates ($key\_set$) and the instances ($I_C$) of certain categories ($C$) (e.g., person, publication, etc.) from G. Then, for each predicate $key \in key\_set$, the algorithm retrieves all the object values of the key for instances in $I_C$. Next, it computes three metrics: discriminability, coverage as shown in Equations 2 and 3 respectively and a F1-score ($F_L$) on them.

$$dis(key, I_C, G) = \frac{|\{o|t =< i, key, o > \in G \wedge i \in I_C\}|}{|\{t|t =< i, key, o > \in G \wedge i \in I_C\}|} \tag{2}$$

$$cov(key, I_C, G) = \frac{|\{i|t =< i, key, o > \in G \wedge i \in I_C\}|}{|I_C|} \tag{3}$$

Note, $i$ and $o$ represent the subject and object of a triple respectively. In the learning process, we remove low-discriminability predicates. Because the discriminability of a predicate is computed based upon the diversity of its object values, having low-discriminability means that many instances have the same object values on this predicate; therefore, when utilizing such object values to look up similar instances, we will not get a suitable reduction ratio.

If any predicate has an $F_L$ (line 11) higher than the given threshold $\alpha$, the predicate with the highest $F_L$ will be chosen to be the candidate selection key. If none of the keys have an $F_L$ above the threshold $\alpha$, the algorithm combines the predicate that has the highest discriminability with every other predicate to form $|key\_set|$-1 virtual predicates, add them to $key\_set$ and remove the old ones. Furthermore, via the function $update(I_C, key\_set, G)$, for a new key, we concatenate the object values of different predicates in the key for the same instance to

form new triples that use the combined virtual predicate as their predicate and the concatenated object values as their objects. These new triples and predicates are added to $G$. The same procedure can then be applied iteratively.

Worst case Algorithm 1 is exponential in the number of candidate keys because of its two loops; but typically only a few passes are made through the outer loop before the termination criteria is met in our current evaluations. For future work, we will explore how to prune the initial list of candidate keys and reduce the data complexity of functions *dis* and *cov* with sampling techniques.

## 4 Index Based Candidate Selection

With the learned predicates, for each instance, we present how to efficiently look up similar instances and compute their similarity based on the objects of such predicates. One simple approach is to compare the object values of the learned predicates for all pairs of instances, e.g., comparing names for people instances. However, this simple method itself might not even scale for large scale datasets. So, we need a technique that enables efficient look-up for similar instances.

### 4.1 Indexing Ontology Instances

We adopt a traditional technique in Information Retrieval (IR) research, the inverted index, to speed up the look-up process. Many modern IR systems allow us to build separate indexes for different fields. Given an RDF graph $G$ and the datatype properties $PR$ learned by Algorithm 1, we use this feature to build indexes for the learned predicates, each of which has posting lists of instances for each token in that field. For a learned predicate $p \in PR$, we extract tokens from the object values of triples using $p$; for each such token $tk$, we collect all instances that are subjects of at least one triple with predicate $p$ and token $tk$ contained in its object value. With the learned predicates, each instance is associated with tuple(s) in the form of (instance, predicate, value) by using the learned predicates individually. We define a function search($Idx$, $q$, $pred$) that returns the set of instances for which the $pred$ field matches the boolean query $q$ using inverted index $Idx$.

### 4.2 Building Candidate Set

With the index, Algorithm 2 presents our candidate selection process where $t$ is a tuple and $t.v$, $t.p$ and $t.i$ return the value, predicate and instance of $t$ respectively. For each tuple $t$, we issue a Boolean query, the disjunction of its tokenized values, to the index to search for tuples ($results$) with similar values on all predicates comparable to that of $t$. The search process performs an exact match on each query token. $is\_sim(t, t')$ returns true if the similarity between two tuple values is higher than a threshold.

First of all, we look up instances on comparable fields. For example, in one of our datasets used for evaluation, we try to match person instances of both the *citeseer:Person* and the *dblp:Person* classes where the key is the combination

**Algorithm 2.** Candidate_Selection($T$,$Idx$), $T$ is a set of tuples using predicates in the learned key; $Idx$ is an inverted index

1. $candidates \leftarrow \emptyset$
2. **for all** $t \in T$ **do**
3. $\quad query \leftarrow$ *the disjunction of tokens of* $t.v$
4. $\quad results \leftarrow \bigcup_{p \in Comparable(t.p)} search(Idx, query, p)$
5. $\quad$ **for all** $t' \in results$ **do**
6. $\quad\quad$ **if** $is\_sim(t, t')$ **then**
7. $\quad\quad\quad candidates \leftarrow candidates \cup (t.i, t'.i)$
8. **return** $candidates$

of *citeseer:Name* and *foaf:Name*. So, for a tuple, we need to search for similar tuples on both predicates. Assuming we have an alignment ontology where mappings between classes and predicates are provided, two predicates $p$ and $q$ are comparable if the ontology entails $p \sqsubseteq q$ (or vice versa).

To further reduce the size of the candidate set, it would be necessary to adopt a second level similarity measure between a given instance ($i$) and its returned similar instances from the Boolean query. Otherwise, any instance that shares a token with $i$ will be returned. In this paper, we compare three different definitions of the function $is\_sim$. The first one is to directly compare (*direct_comp*) two tuple values (e.g., person names) as shown in Equation 4.

$$String\_Matching(t.v, t'.v) > \delta \tag{4}$$

where $t$ and $t'$ are two tuples; $String\_Matching$ computes the similarity between two strings. If the score is higher than the threshold $\delta$, this pair of instances will be added to the candidate set. Although this might give a good pairwise completeness by setting $\delta$ to be low, it could select a lot of non-coreference pairs. One example is person names. Person names can be expressed in different forms: first name + last name; first initial + last name, etc.; thus, adopting a low $\delta$ could help to give a very good coverage on true matches; however, it may also match people with the same family name and similar given names.

Another choice is to check the percentage of their shared highly similar tokens (*token_sim*) as shown in Equation 5:

$$\frac{|sim\_token(t.v, t'.v)|}{\min(|token\_set(t.v)|, |token\_set(t'.v)|)} > \theta \tag{5}$$

where *token_set* returns the tokens of a string; *sim_token* is defined in Eq. 6:

$$\begin{aligned} &sim\_token(s_i, s_j) = \\ &\{token_i \in token\_set(s_i) | \exists token_j \in token\_set(s_j), \\ &String\_Matching(token_i, token_j) > \delta)\} \end{aligned} \tag{6}$$

where $s_{i/j}$ is a string and $token_{i/j}$ is a token from it. Without loss of generality, we assume that the number of tokens of $s_i$ is no greater than that of $s_j$. The

intuition is that two coreferent instances could share many similar tokens, though the entire strings may not be sufficiently similar on their entirety. One potential problem is that it may take longer to calculate because the selected literal values could be long for some instances types (e.g., publication titles).

Instead of computing token level similarity, we can check how many character level n-grams are shared between two tuple values as computed in Equation 7:

$$\frac{|gram\_set(n,t.v) \bigcap gram\_set(n,t'.v)|}{\min(|gram\_set(n,t.v)|,|gram\_set(n,t'.v)|)} > \theta \tag{7}$$

where $gram\_set(n,t.v)$ extracts the character level n-grams from a string. We hypothesize that the n-gram based similarity measure is the best choice. The intuition is that we can achieve a good coverage on true matches to the Boolean query by examining the n-grams (which are finer grained than both tokens and entire strings) while at the same time effectively reducing the candidate set size by setting an appropriate threshold. We use *min* in the denominator for Equations 5 and 7 to reduce the chance of missing true matches due to missing tokens, spelling variations or misspellings (e.g., some tokens of people names can be missing or spelled differently). When building/querying the index and comparing the literal values, we filter stopwords, use lowercase for all characters and ignore the ordering of the tokens and n-grams.

## 5 Evaluation

Our system is implemented in Java and we conducted experiments on a Sun Workstation with an 8-core Intel Xeon 2.93GHz processor and 6GB memory.

### 5.1 Datasets

We evaluate our n-gram based approach on 2 RDF datasets: RKB[3] [8] and SWAT[4]. For RKB, we use 8 subsets of it: ACM, DBLP, CiteSeer, EPrints, IEEE, LAAS-CNRS, Newcastle and ECS. The SWAT dataset consists of RDF data parsed from downloaded XML files of CiteSeer and DBLP. Both datasets describe publications and share some information; but they use different ontologies, and thus different predicates are involved. Their coverage of publications is also different. We compare on 3 instance categories: RKB Person, RKB Publication and SWAT Person. The groundtruth was provided as *owl:sameAs* statements that can be crawled from RKB and downloaded from SWAT as an RDF dump respectively. Since the provided groundtruth was automatically derived and was incomplete and erroneous, we randomly chose 100K instances for each category, applied our algorithm with different thresholds to get candidate selection results, and manually checked the false positives/negatives to verify and augment the groundtruth to improve their quality. We are in the process of completing SWAT Publication groundtruth and will conduct relevant experiments for future work.

[3] http://www.rkbexplorer.com/data/
[4] http://swat.cse.lehigh.edu/resources/data/

We also evaluate on 3 other structured datasets frequently used for evaluating entity coreference systems. Each dataset has a pre-defined schema with several attributes: name, date, etc. We convert them into RDF by treating each attribute as a datatype property. The first one is the Restaurant dataset [16], matching segmented online posts (records) from Fodors (331 records) to Zagat (533 records) with 112 duplicates. It has 4 attributes: name, address, type and city. Another dataset is the Hotel dataset [13] that has 5 attributes: name, rating, area, price and date, matching 1,125 online hotel bidding posts from the Bidding For Travel website[5] to another 132 hotel information records from the Bidding For Travel hotel guides with 1,028 coreferent pairs. The last one is *dataset4* [9], a synthetic census dataset, with 10K records and 5K duplicates within themselves. We remove the Social Security Number from it as was done in BSL [12] to perform a fair comparison and match the 10K records to themselves.

### 5.2 Evaluation Methods and Metrics

In this paper, we adopt a two-phase approach for evaluating our proposed candidate selection algorithm. In phase one, we use the 3 well adopted metrics $PC$, $RR$ and $F_{cs}$ from previous works [21,6] as discussed in Section 2. For phase two, we adopt an actual entity coreference algorithm for detecting *owl:sameAs* links between ontology instances [15] that measures the similarity of two instances by utilizing the triples in an RDF graph as context information. Not only does this context include the direct triples but also triples two steps away from an instance. We apply our candidate selection technique on the RDF datasets discussed in the previous section to select candidate pairs and run this algorithm on the candidate sets to get the F-score of the coreference phase and the runtime of the entire process, including indexing, candidate selection and coreference.

As for parameter settings, for the learning process (Algorithm 1), there are two parameters $\alpha$, determining if a key could be used for candidate selection and $\beta$, determining if a key should be removed. To show the domain independence of our algorithm, we set them to be 0.9 and 0.3 respectively for all experiments. We tested our algorithm on different $\alpha$ and $\beta$ values and it is relatively insensitive to $\beta$, but requires high values for $\alpha$ for good performance. When $\beta$ is low, only a few predicates are removed for not being discriminating enough; when $\alpha$ is high, then we only select keys that discriminate well and are used by most of the instances. For Algorithm 2, different similarity measures may use different parameters. For Equations 4, 5, 6 and 7, we set $\theta$ to be 0.8; for *direct_comp* and *token_sim*, we varied $\delta$ from 0.1 to 0.9 and report the best results. We extract bigrams and compute Jaccard similarity for string matching in all experiments.

### 5.3 Evaluation Results on RDF Datasets

From Algorithm 1, we learned the key for each RDF dataset as following:

[5] www.BiddingForTravel.com

**RKB Person**: full-name, job, email, web-addr and phone
**RKB Publication**: title
**SWAT Person**: citeseer:name and foaf:name

For RKB Person, *full-name* has good coverage but is not sufficiently discriminating; while the other selected predicates have good discriminability but poor coverage. So, they were combined to be the key. For SWAT Person, neither of the two selected predicates has sufficient coverage; thus both were selected.

We compare our method *bigram* (Eq. 7) to *direct_comp* (Eq. 4) and *token_sim* (Eq. 5) that use different string similarity measures; we also compare to *All-Pairs* [2], *PP-Join*(+) [20] and *Ed-Join* [19]; lastly, we compare to *Naive* [15] that detects *owl:sameAs* links without candidate selection. Since *Ed-Join* is not compatible with our Sun machine, we run it on a Linux machine (dual-core 2GHz processor and 4GB memory), and estimate its runtime on the Sun machine by examining runtime difference of *bigram* on the two machines. For coreference results, we report a system's best F-Score from threshold 0.1-0.9. We split each 100K dataset into 10 non-overlapping and equal-sized subsets, index each subset, run all algorithms on the same input and report the average. We conduct a two-tailed t-test to test the statistical significance on the results of the 10 subsets from two systems. On average, there are 6,096, 4,743 and 684 coreferent pairs for each subset of RKB Person, RKB Publication and SWAT Person respectively.

The results are shown in Table 1. Comparing within our own alternatives, for

**Table 1.** Candidate Selection Results on RDF Datasets $|Pairs|$: candidate set size; RR: Reduction Ratio; PC: Pairwise Completeness; $F_{cs}$: the F1-score for RR and PC; F-Score: the F1-Score of Precision and Recall for the coreference results; Total: the runtime for the entire entity coreference process

| Dataset | System | *Candidate Selection* | | | | | *Coref* | *Total* (s) |
|---|---|---|---|---|---|---|---|---|
| | | $\|Pairs\|$ | $RR(\%)$ | $PC(\%)$ | $F_{cs}(\%)$ | $Time(s)$ | *F-score* (%) | |
| RKB Per | bigram (Eq. 7) | 14,024 | **99.97** | 99.33 | 99.65 | 13.32 | **93.48** | **25.45** |
| | direct_comp (Eq. 4) | 104,755 | 99.79 | **99.82** | **99.80** | 14.00 | 92.55 | 51.04 |
| | token_sim (Eq. 5) | **13,156** | **99.97** | 98.52 | 99.24 | 15.72 | 93.37 | 27.13 |
| | All-Pairs [2] | 680,403 | 98.64 | 99.76 | 99.20 | **1.34** | 92.04 | 195.37 |
| | PP-Join [20] | 680,403 | 98.64 | 99.76 | 99.20 | 1.36 | 92.04 | 195.38 |
| | PP-Join+ [20] | 680,403 | 98.64 | 99.76 | 99.20 | 1.39 | 92.04 | 195.42 |
| | Ed-Join [19] | 150,074 | 99.70 | 99.72 | 99.71 | 1.73 | 92.38 | 72.79 |
| | Naive [15] | N/A | N/A | N/A | N/A | N/A | 91.64 | 4,765.46 |
| RKB Pub | bigram (Eq. 7) | 6,831 | **99.99** | **99.97** | **99.98** | 18.26 | **99.74** | **31.73** |
| | direct_comp (Eq. 4) | 7,880 | 99.98 | **99.97** | 99.97 | 22.23 | 99.68 | 36.74 |
| | token_sim (Eq. 5) | **5,028** | **99.99** | 99.80 | 99.89 | 79.91 | 99.70 | 88.96 |
| | All-Pairs [2] | 1,527,656 | 96.94 | 97.95 | 97.44 | 3.93 | 98.59 | 877.80 |
| | PP-Join [20] | 1,527,656 | 96.94 | 97.95 | 97.44 | **3.79** | 98.59 | 877.66 |
| | PP-Join+ [20] | 1,527,656 | 96.94 | 97.95 | 97.44 | 4.00 | 98.59 | 877.87 |
| | Ed-Join [19] | 2,579,333 | 94.84 | 98.57 | 96.67 | 409.08 | 99.04 | 1,473.47 |
| | Naive [15] | N/A | N/A | N/A | N/A | N/A | 99.55 | 34,566.73 |
| SWAT Per | bigram (Eq. 7) | 7,129 | **99.99** | 98.72 | 99.35 | 13.46 | **95.07** | **21.21** |
| | direct_comp (Eq. 4) | 90,032 | 99.82 | 99.86 | **99.84** | 14.30 | 95.06 | 51.33 |
| | token_sim (Eq. 5) | **6,266** | **99.99** | 96.81 | 98.37 | 16.58 | **95.07** | 23.70 |
| | All-Pairs [2] | 508,505 | 98.98 | **99.91** | 99.44 | **1.00** | 95.06 | 108.89 |
| | PP-Join [20] | 508,505 | 98.98 | **99.91** | 99.44 | 1.01 | 95.06 | 108.90 |
| | PP-Join+ [20] | 508,505 | 98.98 | **99.91** | 99.44 | 1.04 | 95.06 | 108.92 |
| | Ed-Join [19] | 228,830 | 99.54 | 99.79 | 99.66 | 1.48 | 95.01 | 51.66 |
| | Naive [15] | N/A | N/A | N/A | N/A | N/A | 95.02 | 12,139.60 |

all datasets, both *bigram* and *token_sim* have the best *RR* while *direct_comp* commonly has better *PC*. *bigram* selected almost as few pairs as *token_sim*, and always has better *PC*.

On RKB Person, *bigram*'s $F_{cs}$ was not as good as that of *direct_comp* and *Ed-Join*; statistically, the difference between *bigram* and *direct_comp* is significant with a P value of 0.0106; the difference between *bigram* and *token_sim* and *All-Pairs/PP-Join*(+) is statistically significant with P values of 0.0004 and 0.0001 respectively. Also, by applying our entity coreference system to the selected pairs, *bigram* has the best F-score that is statistically significant compared to that of *All-Pairs/PP-Join*(+) with a P value of 0.0093.

We observed similar results on SWAT Person. For $F_{cs}$, the difference between *bigram* to *direct_comp*, *token_sim* and *Ed-Join* is statistically significant with P values of 0.0011, 0.0001 and 0.0263 respectively but not to *All-Pairs/PP-Join*(+). Similarly, all other systems took longer to finish the entire process than *bigram*. As for the *F-score* of the coreference results, we didn't observe any significant difference among the different systems.

On RKB Publication, *bigram* dominates the others in all aspects except for |*pairs*|. For $F_{cs}$, except for *direct_comp*, the difference between *bigram* and others is statistically significant with P values of 0.0001. As for the coreference results, although no statistical significance was observed between *bigram* and *direct_comp*/*token_sim*, statistically, *bigram* achieved a better F-score than *All-Pairs/PP-Join*(+)/*Ed-Join* with P values of 0.0001. Similarly, adopting *bigram* gave the best runtime.

Note that *token_sim* took longer to finish than *bigram* even with fewer selected pairs because it took longer to select candidate pairs. It would potentially have to compare every pair of tokens from two strings, which was time-consuming. This was even more apparent on RKB Publication where titles generally have more tokens than people names do.

Finally, we ran our coreference algorithm (*Naive*) on the subsets of each RDF dataset. Although our proposed candidate selection algorithms were typically slower than their competitors, they filtered out many more pairs, which led to faster times for the complete system. Table 1 shows that using *bigram* was the fastest of all options; it was 169.08, 529.65 and 938.30 times faster than *Naive* on RKB Person, SWAT Person and RKB Publication; and by applying candidate selection, the F-score of the coreference results doesn't drop and even noticeably better performance was achieved. For RKB Person and RKB Publication, the improvement on the F-score is statistically significant with P values of 0.0020 and 0.0005. Such improvement comes from better precision: by only comparing the disambiguating information selected by Algorithm 1, candidate selection filtered out some false positives that could have been returned as coreferent by *Naive*. E.g., *Naive* might produce a false positive for RKB Person for two frequent co-authors, because the titles and venues of their papers are often the same; however, by only considering their most disambiguating information, they could be filtered out. In this case, candidate selection doesn't only help to scale the entire entity coreference process but also improves its overall F-score.

To further demonstrate the capability of our technique (*bigram*) in scaling entity coreference systems, we run *Naive* with and without it on up to 20K instances from each of the RDF datasets respectively and measure the speedup factor, computed as the runtime without *bigram* divided by the runtime with it, as shown in Figure 1. The runtime includes both the time for candidate selection

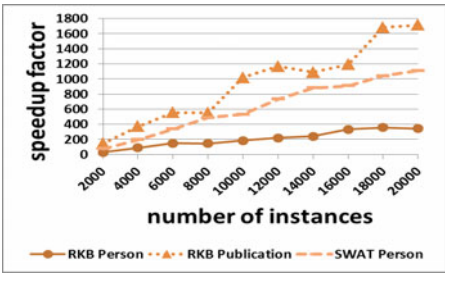


**Fig. 1.** Runtime Speedup by Applying Candidate Selection

and entity coreference. The entire coreference process was speeded up by 2 to 3 orders of magnitude. RKB Person shows less speedup than the others: first, candidate selection found more pairs for RKB Person; second, RKB Person has fewer paths than the other datasets, so there is less to prune.

### 5.4 Evaluation Results Using Standard Coreference Datasets

To show the generality of our proposed algorithm, we also evaluate it on three non-RDF but structured datasets frequently used for evaluating entity coreference algorithms: Restaurant, Hotel and Census as described earlier. We learned the candidate selection key for each dataset as following:

**Restaurant**: name
**Hotel**: name
**Census**: date-of-birth, surname and address_1

Here, we compare to five more systems: BSL [12], ASN [21], Marlin [3], AF [9] and Best Five [18] by referencing their published results. We were unable to obtain executables for these systems.

Here, we apply candidate selection on each of the three full datasets. First, the scale of the datasets and their groundtruth is small. Also, each of the Restaurant and the Hotel datasets is actually composed of two subsets and the entity coreference task is to map records from one to the other; while for other datasets, we detect coreferent instances within all the instances of a dataset itself. So, it is difficult to split such datasets. We didn't apply any actual coreference systems to the candidate set here due to the small scale and the fact that we couldn't run some of the systems to collect the needed candidate sets. Instead, in order to accurately reflect the impact of $RR$, we suggest a new metric $RR_{log}$ computed

as $1 - \frac{\log |candidate\ set|}{\log (N*M)}$. In Table 1, on RKB Person, an order of magnitude difference in detected pairs between *bigram* and *Ed-Join* is only represented by less than 1 point in $RR$; however, a more significant difference in the total runtime was observed. With this new metric, *bigram* and *Ed-Join* have an $RR_{log}$ of 46.13% and 32.77% respectively where the difference is now better represented by 13.36%. We also compute a corresponding $F_{cs\_log}$ using $RR_{log}$. For systems where we reused reported results, we calculated $|pairs|$ from their reported $RR$; because BSL is supervised (thus the blocking was not done on the full dataset), we assumed the same $RR$ as if it was done on the full dataset.

Table 2 shows the results. Since not all systems reported results on all datasets, we only report the available results here. Comparing within our own alternatives,

**Table 2.** Candidate Selection Results on Standard Coreference Datasets

| Dataset | System | *Candidate Selection* | | | | | |
|---|---|---|---|---|---|---|---|
| | | $\|Pairs\|$ | $RR(\%)$ | $PC(\%)$ | $F_{cs}(\%)$ | $RR\_log(\%)$ | $F_{cs\_log}(\%)$ |
| Restaurant | bigram (Eq. 7) | **182** | **99.90** | 98.21 | 99.05 | **56.92** | **72.07** |
| | direct_comp (Eq. 4) | 2,405 | 98.64 | **100.00** | **99.31** | 35.56 | 52.46 |
| | token_sim (Eq. 5) | 184 | **99.90** | 95.54 | 97.67 | 56.83 | 71.27 |
| | All-Pairs [2] | 1,967 | 98.89 | 99.11 | 99.00 | 37.22 | 54.12 |
| | PP-Join [20] | 1,967 | 98.89 | 99.11 | 99.00 | 37.22 | 54.12 |
| | PP-Join+ [20] | 1,967 | 98.89 | 99.11 | 99.00 | 37.22 | 54.12 |
| | Ed-Join [19] | 6,715 | 96.19 | 96.43 | 96.31 | 27.06 | 42.26 |
| | BSL [12] | 1,306 | 99.26 | 98.16 | 98.71 | 40.61 | 57.45 |
| | ASN [21] | N/A | N/A | <96 | <98 | N/A | N/A |
| | Marlin [3] | 78,773 | 55.35 | **100.00** | 71.26 | 6.67 | 12.51 |
| Hotel | bigram (Eq. 7) | **4,142** | **97.21** | 94.26 | **95.71** | **30.06** | **45.58** |
| | direct_comp (Eq. 4) | 10,036 | 93.24 | 96.69 | 94.94 | 22.63 | 36.67 |
| | token_sim (Eq. 5) | 4,149 | **97.21** | 90.56 | 93.77 | 30.04 | 45.12 |
| | All-Pairs [2] | 6,953 | 95.32 | 95.91 | 95.62 | 25.71 | 40.55 |
| | PP-Join [20] | 6,953 | 95.32 | 95.91 | 95.62 | 25.71 | 40.55 |
| | PP-Join+ [20] | 6,953 | 95.32 | 95.91 | 95.62 | 25.71 | 40.55 |
| | Ed-Join [19] | 17,623 | 88.13 | 98.93 | 93.22 | 17.90 | 30.31 |
| | BSL [12] | 27,383 | 81.56 | **99.79** | 89.76 | 14.20 | 24.86 |
| Census | bigram (Eq. 7) | 166,844 | 99.67 | 97.76 | 98.70 | 32.17 | 48.41 |
| | direct_comp (Eq. 4) | 738,945 | 98.52 | 98.08 | 98.30 | 23.77 | 38.27 |
| | token_sim (Eq. 5) | 163,207 | 99.67 | 96.36 | 97.99 | 32.30 | 48.38 |
| | All-Pairs [2] | **5,231** | **99.99** | **100.00** | **99.99** | **51.70** | **68.16** |
| | PP-Join [20] | **5,231** | **99.99** | **100.00** | **99.99** | **51.70** | **68.16** |
| | PP-Join+ [20] | **5,231** | **99.99** | **100.00** | **99.99** | **51.70** | **68.16** |
| | Ed-Join [19] | 11,010 | 99.98 | 99.50 | 99.74 | 47.50 | 64.30 |
| | AF [9] | 49,995 | 99.9 | 92.7 | 96.17 | 38.97 | 54.87 |
| | BSL [12] | 939,906 | 98.12 | 99.85 | 98.98 | 22.42 | 36.62 |
| | Best Five [18] | 239,976 | 99.52 | 99.16 | 99.34 | 30.12 | 46.21 |

for all datasets, *direct_comp* has the best $PC$; *bigram* and *token_sim* have identical $RR$, but *bigram* always has better $PC$. Furthermore, *bigram* always has the best $F_{cs\_log}$ and has better $RR_{log}$ on Restaurant and Hotel but only slightly worse on Census than *token_sim*.

Compared to other systems, on both Restaurant and Hotel, *bigram* has the best $RR$, $F_{cs}$, $RR_{log}$ and $F_{cs\_log}$, though its $F_{cs\_log}$ was only slightly better than that of *All-Pairs*/*PP-Join*(+). Also, with better $RR$, it only has slightly worse $PC$ than *All-Pairs*/*PP-Join*(+)/Marlin on Restaurant. Particularly, *bigram* has significantly better $RR$ (15.65% and 9.08% higher) than $BSL$ and *Ed-Join*

on Hotel; however it was not able to achieve a $PC$ as good as these two systems did. If we consider larger datasets, such a significant difference in $RR$ may save a great amount of runtime. Note that with the two new metrics, the impact of the number of selected pairs becomes more apparent, which we believe more accurately reflects its impact. On Census, $All\text{-}Pairs/PP\text{-}Join(+)$ achieved the best $F_{cs}$ and $F_{cs\_log}$; while *bigram* still achieved better $RR$ than $BSL$ and $BestFive$ with slightly worse $PC$. *bigram* only has a $PC$ of 97.76% because our method only performs exact look-ups into the index; however, in this synthetic dataset, coreferent records were generated by modifying the original records, including adding misspellings, removing white spaces, etc. Therefore, some of the coreferent records couldn't even be looked up. In future work, we will explore techniques for efficient fuzzy retrieval to overcome this problem.

### 5.5 Scalability of Candidate Selection

Figure 2 presents the runtime by applying *bigram* on up to 1 million instances of RKB Person, RKB Publication, SWAT Person and Census, showing that it scales well on large scale datasets. Due to limited availability of high quality groundtruth, we only measured the runtime. For SWAT Person, there are only 500K instances in the dataset. *bigram* scales better on RKB Person since few instances actually use the selected predicates other than *full-name*. Note that

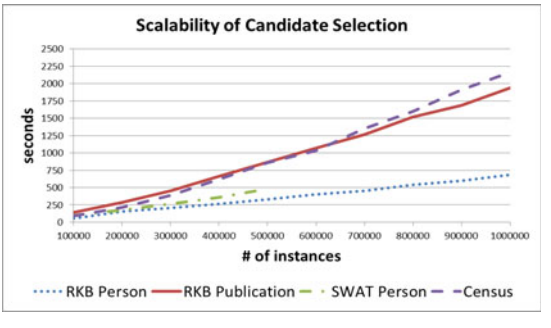


**Fig. 2.** Scalability of the Proposed Candidate Selection Algorithm

All-Pairs/PP-Join(+) couldn't scale to 200K instances on any of these datasets due to insufficient memory, though they fared significantly better $F_{cs}$ on Census.

### 5.6 Discussion

One limitation of our algorithm is that it currently targets datasets that are primarily composed of strings, and we adopt the same string similarity measure for numeric values, e.g., telephone number. Given that a lot of telephone numbers could be very similar to each other, counting the shared bigrams between two such numbers might greatly increase candidate set size, particularly when the data is primarily describing instances in the same geographic area.

Another problem is that we currently perform exact match on each query token when looking up similar instances with the index. This should work well on datasets with decent data quality; however, when there are a lot of errors (e.g., misspellings, missing characters or tokens, etc.), our algorithm may not even be able to retrieve all coreferent instances for a given instance. One possible solution to this problem is to adopt fuzzy matching. We could compute the Soundex code for each token and tokens with the same code are treated *similar*. For a given token, we query the index with all its similar tokens.

Finally, although *bigram* was only tested on 1 million instances (which is relatively small compared to the entire Linked data), it is larger than the number of instances in many Linked Data sets. Also, the number of instances is much smaller than the number of triples (e.g., DBPedia has 672 million triples but only 3.5 million instances), and we perform an initial filtering that instances must be of *comparable types*. Assuming around 100 million instances exist in Linked Data, they could be conservatively grouped into at least 10 sets of *comparable types* with no more than 10 million instances each. Extrapolating from Figure 2, our candidate selection could be computed in about 5.5 hours for each.

## 6 Conclusion

In this paper, we present an index based domain-independent candidate selection algorithm for scalably detecting *owl:sameAs* links. We learn a set of predicates for candidate selection through unsupervised learning. By indexing the instances on the learned predicates' object values, our algorithm is able to efficiently look up similar instances. In the author, publication, restaurant, hotel and census domains, using a bigram-based similarity measure, our algorithm almost always had a better *RR* than all alternatives, and when a full entity coreference algorithm was applied to the results, it led to the best F-score. As a result of its high *RR*, it frequently runs the fastest. Interestingly, our technique enables the overall system to produce coreference results with better F1-score by filtering out possible false positives when comparing only on the most disambiguating information. In the future, we will apply our technique to other entity coreference systems (e.g., [11]) to verify its capability of scaling those systems and improving their overall performances. Also, instead of doing exact lookup into the index, we are interested in exploring methods for efficient fuzzy retrieval.

## References

1. Aswani, N., Bontcheva, K., Cunningham, H.: Mining Information for Instance Unification. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 329–342. Springer, Heidelberg (2006)
2. Bayardo, R.J., Ma, Y., Srikant, R.: Scaling up all pairs similarity search. In: Proceedings of the 16th International Conference on World Wide Web, pp. 131–140 (2007)
3. Bilenko, M., Mooney, R.J.: Adaptive duplicate detection using learnable string similarity measures. In: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 39–48 (2003)

4. Bizer, C., Heath, T., Berners-Lee, T.: Linked data - the story so far. Int. J. Semantic Web Inf. Syst. 5(3), 1–22 (2009)
5. Cao, Y., Chen, Z., Zhu, J., Yue, P., Lin, C.Y., Yu, Y.: Leveraging unlabeled data to scale blocking for record linkage. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI (2011)
6. Elfeky, M.G., Elmagarmid, A.K., Verykios, V.S.: Tailor: A record linkage tool box. In: Proceedings of the 18th International Conference on Data Engineering (ICDE), pp. 17–28 (2002)
7. Euzenat, J., Ferrara, A., Meilicke, C., Nikolov, A., Pane, J., Scharffe, F., Shvaiko, P., Stuckenschmidt, H., Svb-Zamazal, O., Svtek, V., Trojahn dos Santos, C.: Results of the ontology alignment evaluation initiative 2010. In: Proceedings of the 4th International Workshop on Ontology Matching (2010)
8. Glaser, H., Millard, I., Jaffri, A.: RKBExplorer.com: A Knowledge Driven Infrastructure for Linked Data Providers. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 797–801. Springer, Heidelberg (2008)
9. Gu, L., Baxter, R.A.: Adaptive filtering for efficient record linkage. In: Proceedings of the Fourth SIAM International Conference on Data Mining (2004)
10. Hassell, J., Aleman-Meza, B., Arpinar, I.B.: Ontology-Driven Automatic Entity Disambiguation in Unstructured Text. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 44–57. Springer, Heidelberg (2006)
11. Hu, W., Chen, J., Qu, Y.: A self-training approach for resolving object coreference on the semantic web. In: Proceedings of the 20th International Conference on World Wide Web (WWW), pp. 87–96 (2011)
12. Michelson, M., Knoblock, C.A.: Learning blocking schemes for record linkage. In: The Twenty-First National Conference on Artificial Intelligence, AAAI (2006)
13. Michelson, M., Knoblock, C.A.: Creating relational data from unstructured and ungrammatical data sources. J. Artif. Intell. Res. 31, 543–590 (2008)
14. Sleeman, J., Finin, T.: Computing FOAF co-reference relations with rules and machine learning. In: Third International Workshop on Social Data on the Web (2010)
15. Song, D., Heflin, J.: Domain-independent entity coreference in RDF graphs. In: Proceedings of the 19th ACM Conference on Information and Knowledge Management (CIKM), pp. 1821–1824 (2010)
16. Tejada, S., Knoblock, C.A., Minton, S.: Learning domain-independent string transformation weights for high accuracy object identification. In: Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 350–359 (2002)
17. Volz, J., Bizer, C., Gaedke, M., Kobilarov, G.: Discovering and Maintaining Links on the Web of Data. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 650–665. Springer, Heidelberg (2009)
18. Winkler, W.E.: Approximate string comparator search strategies for very large administrative lists. Tech. rep., Statistical Research Division, U.S. Census Bureau (2005)
19. Xiao, C., Wang, W., Lin, X.: Ed-join: an efficient algorithm for similarity joins with edit distance constraints. Proc. VLDB Endow. 1(1), 933–944 (2008)
20. Xiao, C., Wang, W., Lin, X., Yu, J.X.: Efficient similarity joins for near duplicate detection. In: Proceedings of the 17th International Conference on World Wide Web (WWW), pp. 131–140 (2008)
21. Yan, S., Lee, D., Kan, M.Y., Giles, C.L.: Adaptive sorted neighborhood methods for efficient record linkage. In: ACM/IEEE Joint Conference on Digital Libraries (JCDL), pp. 185–194 (2007)