Heuristic-Based Configuration Learning for Linked Data Instance Matching

Khai Nguyen[1,2(✉)] and Ryutaro Ichise[1,2]

[1] The Graduate University for Advanced Studies, Hayama, Japan
{nhkhai,ichise}@nii.ac.jp
[2] National Institute of Informatics, Tokyo, Japan

Abstract. Instance matching in linked data has become increasingly important because of the rapid development of linked data. The goal of instance matching is to detect co-referent instances that refer to the same real-world objects. In order to realize such instances, instance matching systems use a configuration, which specifies the matching properties, similarity measures, and other settings of the matching process. For different repositories, the configuration is varied to adapt with the particular characteristics of the input. Therefore, the automation of configuration creation is very important. In this paper, we propose *cLink*, a supervised instance matching system for linked data. *cLink* is enhanced by a heuristic algorithm that learns the optimal configuration on the basic of input repositories. We show that *cLink* can achieve effective performance even when being given only a small amount of training data. Compared to previous configuration learning algorithms, our algorithm significantly improves the results. Compared to the recent supervised systems, *cLink* is also consistently better on all tested datasets.

Keywords: Instance matching · Schema-independent · Supervised · Linked data

1 Introduction

The general problem of instance matching was first employed in the 1950s. Up to now, researchers have conducted numerous related studies. The targets of the solutions for this problem are diverse, from the general areas of data mining, to more specific application of data cleansing and integration. Instance matching is very important in integrating information from multiple sources because it maintains the consistency and integrity of the final data. This study focuses on instance matching for linked data repositories. Instance matching on linked data has been a topic of extensive research [3]. However, developing a perfect instance matching algorithm remains an open research problem.

The main difficulties of instance matching originate from the ambiguity and heterogeneity of input repositories. The ambiguity exists because many instances of a repository can share similar descriptions (e.g., *Tokyo bay* and *Tokyo station*), especially when the number of instances is high. Meanwhile, the heterogeneity

G. Qi et al. (Eds.): JIST 2015, LNCS 9544, pp. 56–72, 2016.
DOI: 10.1007/978-3-319-31676-5_4

relates to the inconsistent representations of the identical information, including the instance-level (e.g., *Tokyo* and *the capital of Japan*) and schema-level[1] (e.g., *name* and *label*). Compared to other type of data, web-based data like linked data is more heterogeneous and ambiguous due to its high scale and the open-access mechanism.

Basically, in order to match two instances of different repositories, instance matching algorithms have to estimate the similarities between the values of the same attributes. After that, these similarities are aggregated into one matching score. This score is used to determine whether two instances are co-referent or not. The mechanisms of those steps can be described by a matching configuration [12,25].

The construction of configuration defines the approach of an instance matching system. Basic methods [2,19] manually define the configuration and thus are limited to only the repositories whose schema are well-documented and realized by user. Advanced methods create the matching configuration automatically using the observation on the input repositories. They are also called schema-independent methods. For obtaining the schema-independent goal, many techniques have been proposed, such as pure statistic [1,17], unsupervised learning [18], and supervised learning [6–8,11,13,21]. Among them, the supervised learning though requires a number of labeled co-referent instances, delivers the best accuracy.

Recently, supervised learning of configuration has been investigated with genetic algorithm [8,11] and information-gain based selection [7]. The reported results of using these methods are promising. Unfortunately, the genetic algorithm is not supported by a clear strategy as it is based on random search principle. Meanwhile, the information-gain based selection ignores the evaluation of combining different similarity estimation methods. We are motivated in constructing an instance matching system that can take these issues into account.

We propose *cLink*, an instance matching system that is enhanced by a novel configuration learning algorithm named *cLearn*. From two input repositories, *cLink* performs a pre-learning stage to automatically generate initial similarity functions and co-referent candidates. Given that some candidates are labeled and input into *cLearn*, the algorithm uses a heuristic search method to optimize the combination of similarity functions as well as all other settings of the instance matching process. The learning outcome of *cLearn* is an optimal configuration for discovering the co-references from unlabeled candidates. *cLearn* is previously presented in its initial form in [15]. In this paper, we present the *cLink* system whose core learning algorithm is *cLearn* and report in detail many important descriptions, analyses, and evaluations.

The remaining parts of this paper are organized as follows. Section 2 reviews remarkably related work. Section 3 describes the details of *cLink*. Section 4 reports the experiments. Section 5 summarizes the paper and future work.

[1] In this paper, the schema of a linked data repository is defined as the list of all predicates, which is part of the RDF statements ($< subject, predicate, object >$).

2 Related Work

SILK [25] and LIMES [12] are among the frontiers of linked data instance matching frameworks. These frameworks define a declarative structure to represent a manually-defined matching configuration. Zhishi.Links [19], AggrementMaker [2], and RiMOM [10] are more advanced system of manual approach, which focus on both accuracy and scalability. Zhishi.Links leverages domain knowledge while AggrementMaker and RiMOM combine many strategies to improve matching accuracy. RiMOM is also one of the current state-of-art automatic instance matching systems.

SERIMI [1] and SLINT+ [17] attempted to eliminate user involvement by automatically detecting property alignments using overlap measure. KnoFuss [18] is an unsupervised system that learn the matching configuration using genetic algorithm. The advantage of Knofuss is that this system does not require labeled data for training. However, since the fitness value is calculated using a pseudo value of the actual accuracy, the learned configuration can contain incorrect alignments and thus reduce the performance. PARIS [24] is an automatic system that consider the similarities of RDF objects, predicates, instances, and classes as related components. The matching process is done by iteratively computing and propagating the similarities over these components. Also based on probabilistic theory but combining with crowdsourcing technology, Zencrowd offers a large scale instance matching framework. Although these automatic systems obtained good results in experiment, when the high accuracy is the first priority and the existing co-references are available, the supervised learning approach is still the first option.

ADL [7] and ObjectCoref [6] learn discriminative properties that are expected to effectively separate the positive and negative co-references. These systems include the domain knowledge in order to improve the accuracy. Instance matching also can be treated as a binary classification problem. A benchmark of many different classifiers for linked data instance matching can be found in [23]. Recently, Rong et al. used Adaboost algorithm to enhance the performance of a random forest classifier [21]. The disadvantage of using classifier is that the matching scores of instances are not explicitly calculated and hence the further post-processing steps, whose these scores are the demand, cannot be implemented. For example, adaptive filtering in SLINT+ greatly improves the result [17].

Most related to our work are RAVEN [11], EAGLE [13], EUCLID [14], and ActiveGenLink [8], which learn the matching configuration using supervised approach. While RAVEN and EUCLID use deterministic methods, EAGLE and ActiveGenLink apply genetic algorithm in order to improve the efficiency, especially when many attributes are input. However, genetic algorithm still has to check many configurations to reach the convergence. In addition, active learning is used for RAVEN, EAGLE, and ActiveGenLink. Compared to these systems, *cLink* is different at the novel learning algorithm. *cLink* also supports many similarity aggregation strategies and matching score post-filtering. In the next section, we describe the detail of *cLink*.

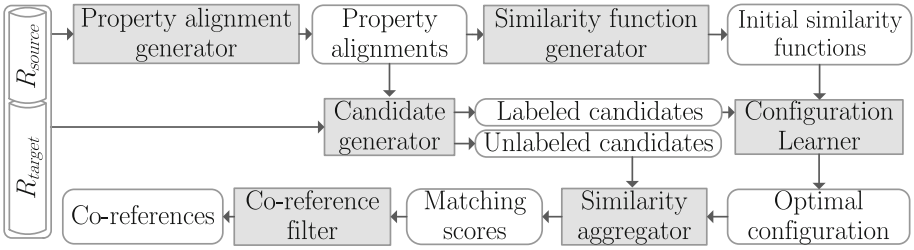

Fig. 1. The workflow of *cLink*.

3 The *cLink* System

cLink consists of six components, property alignment generator, similarity function generator, candidate generator, configuration learner, similarity aggregator, and co-reference filter. The workflow of *cLink* is illustrated in Fig. 1. Given two input repositories, R_{source} and R_{target}, *cLink* first creates the property alignments. Then, these alignments are used to build the initial similarity functions and to select the possibly co-referent candidates. Using the existing co-references, some candidates are labeled and input into the configuration learner to find an optimal configuration. The optimal configuration defines the appropriate similarity functions selected from the initial similarity functions. Each similarity function computes the similarity of two instances on one property. The similarity aggregator combines many similarities into a final matching score. Finally, co-references filter considers the matching score of all candidates and produces the final co-references. Next, we describe the details of each component.

3.1 Property Alignment Generator

This component generates the property alignments between R_{source} and R_{target}. An appropriate property alignment is expected to describe the same information of two instances (e.g., 'born in' and 'home town'). In linked data, as properties are represented by RDF predicates, the outputs of this component are the alignments between RDF predicates.

Property alignments are generated in two steps. First, *cLink* selects in the source repository the predicates that satisfy two conditions. Second, *cLink* aligns each selected predicate with the corresponding ones in the target repository using an overlap measure. The conditions used in the first step are the discriminability and the coverage, which are constructed by a modification on the basic of [22]. The discriminability $discr(p_k)$ expresses the diversity of the RDF objects declared by p_k, while the coverage $cover(p_k)$ represents the instance-wise frequency of p_k. Eqs. 1 and 2 describe $discr(p_k)$ and $cover(p_k)$, respectively:

$$discr(p_k) = \frac{|\{o|x \in R_{source}, < s, p_k, o > \in x\}|}{|\{< s, p_k, o > |x \in R_{source}, < s, p_k, o > \in x\}|} \quad (1)$$

$$cover(p_k) = \frac{|\{x|x \in R_{source}, < s, p_k, o > \in x\}|}{|R_{source}|} \tag{2}$$

where R stands for repository, $< s, p, o >$ is a RDF triple, and x is an instance, which is a set of RDF triples sharing the same subject s. We separate the predicates by their data type: $string, number, date$, and URI. The type of a predicate is assigned with the most frequent type of its RDF objects. For each type, we select the predicates having the discriminability $discr$ higher than α and then retain only the K_{cover} predicates with the highest coverages $cover$.

One predicate selected from the source repository can be aligned with many predicates having the same type in the target. The confidence $overlap$ of an alignment $[p_{source}, p_{target}]$ is measured as follows:

$$overlap([p_{source}, p_{target}]) = \frac{|O_{p_{source}} \cap O_{p_{target}}|}{|O_{p_{source}}|} \tag{3}$$
$$O_{p_k} = \{E(o)|x \in R_k, < s, p_k, o > \in x\}$$

where E is a pre-processing function. E is applied for each RDF object. E works in flexible ways depending on the property type. For $string$, E collects the tokens. For $number$, E returns the rounded value at two decimal digits. For $date$, E keeps the original value. For URI, E extracts the remaining string after stripping away the domain. For each predicate of the source repository, we select the top K_{align} alignments having the highest $overlap$. The confidence is not enough to guarantee the correctness of generated alignments. Therefore, a medium or large number of property alignments is safer to retain the useful alignments. Removing incorrect alignments is the mission of configuration learner. In *cLink*, we set $\alpha = 0.5, K_{cover} = 4$ and $K_{align} = 4$ for all types as the default parameter. In total, by default this component produces at most 64 property alignments.

Equation 3 works under the assumption that the target repository contains many co-references with the source repository. Therefore, the denominator is related only to the source repository instead of both repositories like a Jaccard measure, which is used in SERIMI [1] and SLINT+ [17]. Equation 3 is reasonable because currently there are many large repositories that cover a wide range of instances (e.g., DBpedia, Freebase, and Wikidata). Furthermore, as the predicate alignments are generated using only RDF objects, *cLink* does not require the specification of the schemata. In other words, *cLink* is schema-independent.

3.2 Similarity Function Generator

This component assigns to each property alignment the suitable similarity metrics and the result of one assignment is one similarity function. In other word, each similarity function computes the similarity of two instances on one designated property. The final matching score is computed by multiple similarity functions. The detail is described in Sect. 3.5. Equation 4 is the definition of a similarity function sim:

$$sim_{\{[p_{source},p_{target}],metric\}}(x,y) = \max_{o_x,o_y}(metric(E(o_x),E(o_y))) \quad (4)$$
$$< s, p_{source}, o_x > \in x, < s, p_{target}, o_y > \in y$$

A similarity function is specified by two pieces of information: a property alignment $[p_{source}, p_{target}]$ and a similarity measure *metric*. For two instances: $x \in R_{source}$ and $y \in R_{target}$, a similarity function returns the similarity of the most similar RDF objects declared by p_{source} and p_{target}. In other words, the max operator effects if p_{source} or p_{target} appears many times in x or y. The function E defined in Sect. 3.1 is used to pre-process the RDF objects.

Table 1. Example of property alignments and assigned similarity measures

Source property	Target property	Similarity measures
Label	name	Levenshtein, TFIDF-Cosine
	leader	Levenshtein, TFIDF-Cosine
Description	comment	TFIDF-Cosine
	abstract	TFIDF-Cosine
Establish	named date	ExactMatch
Population	area size	ReversedDifference
	elevation	ReversedDifference

The similarity measure *metric* is assigned to the similarity function in accordance with the type of p_{source} and p_{target}, as illustrated in Table 1. *cLink* supports five similarity measures. For 'date' and 'URI', *cLink* uses the exact matching, which returns 1.0 if two values are identical and 0.0 otherwise. For 'number', *cLink* uses the reversed difference (Eq. 5), which calculates how much close two numbers a and b are to each other:

$$diff(a,b) = (1 + |a - b|)^{-1} \quad (5)$$

For *string*, *cLink* supports two measures having different characteristics. For short string[2], we select Levenshtein because of its robustness. For long string, we select *cosine* similarity on TFIDF vectors calculated from the token sets. In order to obtain the TFIDF weight for each token, we adapt the original weighting calculation for being compatible with linked data instances. We calculate the normalized term frequency for TF and probabilistic inverse frequency for IDF, which are described by Eqs. 6 and 7, respectively.

$$TF(w,x,R) = \frac{|\{< s,p,o > | w \in E(o), < s,p,o > \in x\}|}{\max_{y \in R} |\{< s,p,o > | w \in E(o), < s,p,o > \in y\}|} \quad (6)$$

$$IDF(w,R) = \log \frac{|R| - |\{y| < s,p,o > \in y, w \in E(o), y \in R\}|}{|\{y| < s,p,o > \in y, w \in E(o), y \in R\}|} \quad (7)$$

[2] The strings whose length is less than 100 characters.

where, w is the token of interest, x is an instance belonging to the repository R. Function E extracts the tokens of the input RDF object.

All property alignments generated previously are input into this component. In parallel, the alignments of string properties are input into the candidate generator, which is described in the following section.

3.3 Candidate Generator

This component detects the pairs of potentially co-referent instances. Such pairs are called candidates. *cLink* uses a simple token-based prefix blocking approach [20], by which a pair of two instances is considered as a candidate if they share a number of first tokens of the RDF objects declared by designated properties. In *cLink*, we use only one token so that *cLink* can retain as many correct candidates as possible. In addition, only the generated string property alignments are used for comparing the tokens.

The mission of the candidate generator is to limit the huge number of pairwise alignments between instances, which is $|R_{source}| \times |R_{target}|$. Therefore, this component is very important because it is impractical to check all pairs of instances, especially when the repositories are large.

There is possibility to use weighting schemes for reducing the number of candidates [9,17,25]. However, such reduction is companied with a drop in the number of correct candidates. For that reason, we recommend using the simple token-based blocking without weighting.

Given that positive and negative labels are assigned (e.g., manually or automatically by using some existing co-references) to some candidates for creating labeled ones, they are divided into training set T and validation set V, which are required by the learning algorithm in the next step.

3.4 Configuration Learner

The input of this component are the labeled candidates, including a training set T and a validation set V; initial similarity functions I_{sim} (Sect. 3.2); and similarity aggregators I_{agg} (Sect. 3.5). The output of this component is the optimal configuration C_{opt} that is most suitable to the input repositories. A configuration specifies the combination of similarity functions F_{sim}, the similarity aggregator Agg, the parameters δ_{sim} associated with each similarity function sim, and the parameter δ of the co-reference filter (Sect. 3.6). For solving the configuration learning problem, we use *cLearn* algorithm. *cLearn* uses a heuristic search method to learn the combination of the similarity functions, the similarity aggregator, and other parameters. The pseudo code of *cLearn* is given in Algorithm 1. Note that in the pseudo code, we use dot ('.') notation as the member accessing function. The $Init$ function creates a configuration by assigning Agg, F_{sim}, and δ with given values. The $Evaluate$ function first executes the similarity aggregator and the co-reference filter specified by a configuration, on a set of candidates. Then, based on the label of those candidates, it computes the performance, $F1$ score. $F1$ is

Algorithm 1. *cLearn*

Input: Training set T, validation set V, integer paramerter K_{top}
list of similarity functions I_{sim}, list of similarity aggregators I_{agg}
Output: Optimal configuration C_{opt}

$C_{agg} \leftarrow \emptyset$
foreach $A \in I_{agg}$ **do**
  $visited \leftarrow \emptyset$
  foreach $sim \in I_{sim}$ **do**
    $c \leftarrow Init(Agg \leftarrow A, F_{sim} \leftarrow sim, \delta \leftarrow 0)$
    $[c.\delta, F1] \leftarrow EvaluateAndAssignThreshold(c, T)$
    $c.\sigma_{sim} \leftarrow c.\delta$
    $visited \leftarrow visited \cup \{[c, F1]\}$
  $candidate \leftarrow TopHighestF1(visited, K_{top})$
  while $candidate \neq \emptyset$ **do**
    $next \leftarrow \emptyset$
    foreach $g \in candidate$ **do**
      foreach $h \in candidate$ **do**
        $c \leftarrow Init(Agg \leftarrow A, F_{sim} \leftarrow g.c.F_{sim} \cup h.c.F_{sim}, \delta \leftarrow 0)$
        $[c.\delta, F1] \leftarrow EvaluateAndAssignThreshold(c, T)$
        $visited \leftarrow visited \cup \{[c, F1]\}$
        if $g.c.F_{sim} \neq h.c.F_{sim}$ **and** $F1 \geq g.F1$ **and** $F1 \geq h.F1$ **then**
          $next \leftarrow next \cup \{[c, F1]\}$
    $candidate \leftarrow next$
  $F1 \leftarrow Evaluate(argmax_{v \in visited}(v.F1).c, V)$
  $C_{agg} \leftarrow C_{agg} \cup \{[c, F1]\}$
$[C_{opt}, F1] \leftarrow argmax_{v \in C_{agg}}(v.F1)$
return C_{opt}

the harmonic mean of the recall rec and the precision pre, which are calculated as follow:

$$rec = \frac{\text{Number of correctly detected co-references}}{\text{Number of actual co-references}} \tag{8}$$

$$pre = \frac{\text{Number of correctly detected co-references}}{\text{Number of all detected co-references}} \tag{9}$$

The *EvaluateAndAssignThreshold* function works similarly to *Evaluate*, but at the same time, it finds a value for δ. This function selects the top N candidates with highest matching score, where N is the number of the actual co-references in L. After that, it assigns the lowest score of the correctly detected co-reference to δ. In *cLearn*, K_{top} controls the maximum size of F_{sim} in the learned configuration and is fixed to 16 by default. This parameter also limits the number of configurations that the algorithm has to check to $2^{K_{top}}$ and thus guarantees a constantly maximum response time.

cLearn begins with each single similarity function and then considers the combinations of those individuals. *cLearn* is based on a greedy enhancement heuristic (line 17), which assumes that the performance of a new combination of similarity functions must not be less than that of the combined components. The heuristic is reasonable because a series of similarity functions that reduces the performance has little possibility of generating a further combination with improvement.

The global optimum of configuration can be found using exhaustive search. However, such method is extremely expensive in term of computational cost and thus has not been used. Some systems try to use genetic search to solve the issue [8,13,18] but this algorithm is still time-consuming and contains many parameters. We use a heuristic search to reduce the complexity and minimize the parameters.

cLearn uses validation set V (line 20) in order to increase the generality of the final configuration C_{opt}. Each iteration controlled by line 2 finds an optimal configuration associated to a similarity aggregator A. Therefore, C_{agg} contains $|I_{agg}|$ configurations that is optimized for training set T. In order to identify the most general configuration, using V, a different set from T, is a better option.

cLearn algorithm is generic because the function *EvaluateAndAssignThreshold* can be replaced by other functions having the similar purpose. Therefore, this algorithm not only works in *cLink*, but also is compatible with any configuration-based matching system [7,12,14,25].

3.5 Similarity Aggregator

The similarity aggregator computes the final matching score for each candidate using the similarity functions F_{sim} and their parameter δ_{sim}, which are specified by the configuration. The computation of the matching score $mScore(x, y)$ for two instances x and y is defined as follows:

$$mScore(x, y) = weight(y) \times combine_{F_{sim}}(x, y) \tag{10}$$

where *weight* is a function weighting the target instance y and *combine* is a similarity combination function. *cLink* provides *non-weighting* and *weighting* versions. For *non-weighting*, $weight(y)$ simply returns 1.0. For *weighting*, the weight is calculated by Eq. 11:

$$weight(y) = \log_{\max_{t \in R_{target}} size(t)} size(y) \tag{11}$$

where $size(y)$ counts the number of RDF triples existing in y. By using Eq. 10, we assume that instances containing many triples are more prioritized. The logarithmic scale is used to reduce the weight of instances whose *size* is particularly large. This weighting method is effective when the target repository is very ambiguous, such as large repositories.

For similarity combination, *cLink* uses the following equation:

$$combine_{F_{sim}}(x,y) = \frac{1}{valid(U_{F_{sim}}(x,y))} \sum_{v \in U_{F_{sim}}(x,y)} v^k \tag{12}$$
$$U_{F_{sim}}(x,y) = \{sim(x,y) | sim(x,y) \geq \sigma_{sim}, sim \in F_{sim}\}$$

where $k \in \{1,2\}$, *valid* is a counting function, and σ_{sim} is the parameter for each similarity function *sim*, which is determined automatically by *cLearn* (line 7 of Algorithm 1). k controls the transformation for each similarity v. When $k = 1$, *combine* acts as a first order aggregation. When $k = 2$, we have a quadratic aggregation. There are also two variations of *valid*, which return the number of elements in $U_{F_{sim}}(x,y)$ and 1.0 always. The difference between these variations is that the latter penalizes the (x,y) pair having similarities $sim(x,y) < \sigma_{sim}$ while the former does not. In addition, *cLink* provides a *restriction* mechanism to enable or disable σ_{sim}. When disabling, all σ_{sim} are set to zero instead of their original value. In total, there are 16 combinations of *weight*, *valid*, k, and *restriction*. Consequently, there are 16 different aggregators supported by *cLink*. All aggregators are used to initialize I_{agg} and let *cLearn* select the best one.

3.6 Co-reference Filter

This component produces the final predictive co-references on the basis of matching scores of all candidates. *cLink* reuses the adaptive filter of in SLINT+ [17]. This filter follows the idea of the stable marriage problem [4]. A pair of instances (x,y) is co-referent if its matching score $mScore(x,y)$, where $x \in R_{source}$ and $y \in R_{target}$, satisfies the conditional statement of Eq. 13:

$$mScore(x,y) \geq \max(\max_{z \in R_{source}} mScore(z,y), \max_{t \in R_{target}} mScore(x,t)) \tag{13}$$

In addition, this component uses a cut-off filter to eliminate the incorrect candidates but satisfying Eq. 13. A threshold δ is used for this task. δ is assigned by the learning algorithm. Only instance pairs whose scores satisfy the condition statement of the filter and threshold δ are promoted to be a final co-reference.

Above we have described the details of *cLink*. The next section reports the experiments and the results.

4 Experiment

We report in total three experiments. The first experiment is to evaluate the candidate generator. The second experiment evaluates learning algorithm *cLearn*. In this experiment, we compare *cLearn* with other configuration learning algorithms. The third experiment compares *cLink* with other systems, including the systems that use supervised learning approach. We use the default parameters for the settings of *cLink* for every experiment. The details of the experiments are described from Sect. 4.3 to 4.5.

cLink in implemented as part of ScSLINT framework [16] and is available at http://ri-www.nii.ac.jp/ScSLINT/.

Because *cLink* uses training data for the learning algorithm, before reporting the experiments, we describe the annotation process, which is compatible not only for *cLink* but also for other learning-based instance matching systems.

4.1 Candidate Labeling and Separation

From the input of source repository R_{source}, a target repository R_{target}, and the actual co-references R, we execute the first three components of *cLink* to obtain the candidates. After that, we annotate the candidates using the actual co-references. Concretely, if a candidate (x, y) appears in R, where $x \in R_{source}$ and $y \in R_{target}$, we assign positive label to (x, y) and candidate label to (x, z), where $y \neq z \in R_{target}$.

When splitting the candidates into different sets, we propose to put all the candidates sharing the source instance into one set. In other words, each separated set does not share any source instance with each other. This separation strategy is used because it is compatible with the practical annotation process. In order to keep the quality of annotated data, the following process can be applied. Given an instance from source repository, a ranked list of instance pairs is created using a simple matching method. Annotators are asked to assign the positive labels for the top ranked pairs. Because the list is sorted, the remaining pairs can be assumed to be negative. By following this manner, such positive label assignments have tiny possibility to be incorrect. That is, the quality of training data is guaranteed. In previous experiments [7,8,13,21], other supervised systems randomly select training examples from the candidates set and thus fail to address the issue of annotation quality in real matching tasks.

For our experiments, we first randomly separate the candidates into two sets using the above strategy. The size of these sets is determined differently for each experiment. The first set is reserved for learning, and the second set (test set) is used for evaluating the performance. After that, the first set is similarly separated into training set T (80 %) and validation set V (20 %). Furthermore, since the source repository frequently contains many instances that are not co-referent with any instance in the target, an arbitrary randomization possibly creates the sets that are drastically different from each other in term of the ratio between the positive and the negative candidates. Therefore, in our experiment, we remain the same ratio r for all separated sets, where r is the actual ratio between the number of the source instances related to at least one co-reference and the ones that do not involve with any co-reference.

4.2 Datasets

We use two the datasets provided by the instance matching track of OAEI 2010 and OAEI 2012, which have been considering to be among the most challenging datasets. There are five subsets (D1 to D5) for OAEI 2010, and seven subsets (D6 to D12) for OAEI 2012. The details of these subsets are given in Table 2.

Table 2. Summary of OAEI 2010 (D1 to D5) and OAEI 2012 (D6 to D12) datasets

ID	Source repository			Target repository			Co-references
	Name	#Instances	#Properties	Name	#Instances	#Properties	
D1	Sider	2,670	10	Drugbank	19,689	118	1,142
D2	Sider	2,670	10	Diseasome	8,149	18	344
D3	Sider	2,670	10	Dailymed	10,002	27	3,225
D4	Sider	2,670	10	DBpedia	4,183,461	45,858	1,449
D5	Dailymed	10,002	27	DBpedia	4,183,461	45,858	2,454
D6	NYT loc	3,837	22	DBpedia	4,183,461	45,858	1,917
D7	NYT org	5,967	20	DBpedia	4,183,461	45,858	1,922
D8	NYT peo	9,944	20	DBpedia	4,183,461	45,858	4,964
D9	NYT loc	3,840	22	Freebase	40,358,162	2,455,627	1,920
D10	NYT org	6,045	20	Freebase	40,358,162	2,455,627	3,001
D11	NYT peo	9,958	20	Freebase	40,358,162	2,455,627	4,979
D12	NYT loc	3,785	22	Geonames	8,514,201	14	1,729

Table 3. Result of candidate generation.

	D1	D2	D3	D4	D5	D6
#cans	5,771	4,258	5,013	482,605	987,856	38,201,823
rec	0.9721	0.9535	0.9939	0.9538	0.9780	0.9718
	D7	D8	D9	D10	D11	D12
#cans	61,702,166	46,942,099	222,686,524	357,365,003	620,073,101	32,161,659
rec	0.9880	0.9970	0.9875	0.9770	0.9912	0.9676

For OAEI 2010, we use the data provided by the OAEI website. The OAEI 2010 dataset contained five repositories related to healthcare domain: Sider, Diseasome, Drugbank, Dailymed, and DBpedia. The OAEI 2012 dataset contains four repositories NYTimes (NYT), DBpedia, Freebase, and Geonames with three domains location (loc), organization (org) and people (peo). For this dataset, we use the dump of NYTimes 2014/02, DBpedia 3.7 English, Freebase 2013/09/03, and Geonames 2014/02. There are a few slight inconsistencies between the ground-truth provided by OAEI 2012 and our downloaded dump data, because of the difference in the release dates. Therefore, we exclude 130 (0.298 %) source instances which are related to such inconsistencies.

We use these datasets because the purpose of our experiments is to know the performance of *cLink* on the real data with large size. Although there are some newer datasets, they are either small, artificial, or focus on the benchmarks with some special targets (e.g., reasoning-based, string distortion, language variation). Therefore, we do not use such datasets. In addition, using OAEI 2010 dataset provides us the opportunity to compare our proposed system with other learning-based systems, which were already tested on this dataset.

Table 4. F1 score of *cLink* when using *cLearn* and other algorithms.

	cLearn	*genetic*	*naive*	*none*
D1	0.9365	**0.9375**	0.8767	0.8137
D2	0.8679	0.8450	**0.8733**	0.8430
D3	**0.8686**	0.7665	0.6741	0.6841
D4	**0.6676**	0.6673	0.6407	0.5535
D5	0.4725	**0.4727**	0.3100	0.2080
D6	**0.8835**	0.8249	0.8289	0.7336
D7	**0.9058**	**0.9058**	0.8594	0.4508
D8	**0.9645**	0.9635	0.9581	0.9506
D9	**0.8781**	**0.8781**	0.8609	0.1934
D10	**0.9116**	0.9089	0.8105	0.2223
D11	0.9465	**0.9477**	0.9325	0.4640
D12	**0.9163**	0.9106	0.8908	0.8852
H.Mean	**0.8191**	0.8022	0.7236	0.4249

4.3 Experiment 1: Candidate Generation

We evaluate the recall of the generated candidates by using Eq. 8. In order to use Eq. 8, we consider all the candidates as the detected co-references. We report this experiment because candidate generation is a very important step of instance matching system. The recall of this step is the upper bound of the final recall of the full instance matching process. In addition, we report the number of generated candidates because it reflects the complexity of further tasks.

Table 3 lists the number of the generated candidates *#cans* and the recall *rec* by each subset. According to this table, the recall is very high although it cannot reach 1.0, even when *cLink* uses the token sharing, a very relaxed constraint. Such a lose constraint leads to high number of candidates, compared to the expected co-references (Table 2), especially on OAEI 2012 dataset (D6 to D12). However, compared to the number of all possible instance pairs between the source and the target repository, for all subsets, more than 99.9 % of pairwise alignments are excluded. Shortly, candidate generator shows its important role as it considerably reduces the complexity of further components, such as the similarity aggregator and the configuration learner.

4.4 Experiment 2: Evaluation of *cLearn* Algorithm

We report the result of *cLink* when using *cLearn* and other alternatives, so that we can simultaneously evaluate the effectiveness of *cLearn* and compare it with other algorithms. We implement two baseline algorithms, non-optimization (*none*) and naive (*naive*). *none* does accept all generated similarity functions without any validation. This approach is similar to that of SLINT+ [17] and

Table 5. F1 score of the compared systems on OAEI 2010.

Training data	System	D1	D2	D3	D4	D5	H.mean
5 %	*cLink*	**0.911**	**0.824**	**0.777**	0.6414	**0.424**	**0.663**
	Adaboost	0.903	0.794	0.733	0.641	0.375	0.628
Varied by subset	*cLink*	**0.894**	**0.829**	**0.722**			**0.799**
	ObjectCoref	0.464	0.743	0.708			0.611
Reference systems							
RiMOM		0.504	0.458	0.629	0.576	0.267	0.445
PARIS		0.649	0.108	0.149	0.502	0.219	0.207

SERIMI [1], which totally rely on the accuracy of property alignment generator. *naive* selects the K_{top} similarity functions that offer the highest $F1$ on training data. For other alternatives, we select genetic algorithm (*genetic*) as the state-of-the-art in configuration learning [8,13]. *genetic* represents the combination of similarity functions by binary array, in which an element indicates the active or inactive status of the associated similarity function. We choose exponential ranking for fitness selection, 0.7 for single point cross-over probability, 0.1 for single point mutation probability, and 50 for the population size.

5-folds cross validation is used for this experiment. We choose cross-validation because this option in turn puts all candidates into training as well as testing.

In Table 4, we report the average F1 scores on each subset of the tested algorithms. According to this table, *cLearn* consistently outperforms other algorithms in term of harmonic mean of all subsets. Although *genetic* is competitive with *cLearn* on some subsets, when considering each fold separately, so that there are 60 test cases for 12 subsets, the paired t-test over all tests at 0.05 significant level shows that *cLearn* is significantly better than *genetic*, as well as all other algorithms. In addition, the efficiency of *cLearn* is much better than *genetic*. *genetic* spends averagely 7,231 s[3] for learning on one subset of OAEI 2012 while *cLearn* only takes 2,977 s[3] (See Footnote 3). The average numbers of configurations that *cLearn* and *genetic* have to check is 137 and 263, respectively. That is, almost 50 % configurations are skipped by using *cLearn* compared to *genetic*. This fact supports the efficiency of using our heuristic against the random convergence principle of *genetic*.

4.5 Experiment 3: Compare *cLink* with Other Systems

We compare *cLink* with two supervised instance matching systems, ObjectCoref [6] and the work in [21], which uses Adaboost to train a classifier. In this paper, we temporarily refer to this work with the name Adaboost. In addition, we also report the result of RiMOM [10] and PARIS [24] for reference. RiMOM and PARIS are the two state-of-the-art systems among non-learning based ones.

[3] Tested on a computer equipped with two Intel E5-2690 CPUs and 256 GB memory.

OAEI 2010 dataset is used for this comparison. According to the reported result of Adaboost and ObjectCoref, Adaboost used 5 % candidates for training on all subsets, while ObjectCoref uses 20 actual co-references, which is equivalent to 2.3 %, 11.6 % and 1.2 % candidates on D1, D2, and D3, respectively[4]. Therefore, we feed the same amount of training data into *cLink*, on each respective subset and for each comparison. The results of *cLink* in this experiment are the average of 10 times repeat.

Table 5 reports the F1 scores of *cLink* and other systems. According to this table, *cLink* consistently outperforms the others. In detail, *cLink* improves the results against ObjectCoref and *cLink* is remarkably better than Adaboost on D3 and D5. These subsets are difficult for other systems because they involve with DailyMed, a repository containing the highest number of co-references inside. The comparison also reveals that *cLink* can achieve promising results by being given a small amount of training data (at most 5 %). Compared to the reference systems, RiMOM and PARIS, all of *cLink*, Adaboost, and ObjectCoref are far better. This fact supports the necessity of supervised resolution systems when effectiveness is the first priority.

5 Conclusion and Future Work

We presented *cLink*, an effective and efficient schema-independent entity resolution system. *cLink* supports many aggregation methods and uses a reasonable filtering technique to refine the final resolution results. The matching configuration is optimized by a novel and efficient learning algorithm using only a small amount of training data. The *cLearn* algorithm significantly improves the result of *cLink* against other algorithms, including the ones that are used by other state-of-the-art systems. *cLink* also achieves better performance compared to recent systems.

In future work, we are interested in experimenting with learning goals that are different from F1 for each particular property (e.g., prioritizing the recall or precision). Since there is labeled data, a learning algorithm for optimizing the candidate generator is also very useful to reduce the candidates. In addition, transfer learning and active learning are the two technologies that are applicable in *cLink* in order to even reduce more the training data.

References

1. Araujo, S., Tran, D.T., DeVries, A., Hidders, J., Schwabe, D.: SERIMI: Class-based disambiguation for effective instance matching over heterogeneous web data. In: 15th ACM SIGMOD Workshop on the Web and Databases, pp. 25–30 (2012)
2. Cruz, I.F., Antonelli, F.P., Stroe, C.: AgreementMaker: efficient matching for large real-world schemas and ontologies. VLDB Endowment **2**, 1586–1589 (2009)
3. Ferrara, A., Nikolov, A., Scharffe, F.: Data linking for the semantic web. Int. J. Semant. Web Inf. Syst. **7**(3), 46–76 (2011)

[4] Only the results on D1, D2, and D3 are available for ObjectCoref [5].

4. Gale, D., Shapley, L.S.: College admissions and the stability of marriage. Am. Math. Mon. **96**(1), 9–15 (1962)
5. Hu, W., Chen, J., Cheng, G., Qu, Y.: ObjectCoref & Falcon-AO: results for OAEI 2010. In: 5th ISWC Workshop on Ontology Matching, pp. 158–165 (2010)
6. Hu, W., Chen, J., Qu, Y.: A self-training approach for resolving object coreference on the semantic web. In: 20th International Conference on World Wide Web, pp. 87–96 (2011)
7. Hu, W., Yang, R., Qu, Y.: Automatically generating data linkages using class-based discriminative properties. Data Knowl. Eng. **91**, 34–51 (2014)
8. Isele, R., Bizer, C.: Active learning of expressive linkage rules using genetic programming. Web Semant.: Sci. Serv. Agents World Wide Web **23**, 2–15 (2013)
9. Isele, R., Jentzsch, A., Bizer, C.: Efficient multidimensional blocking for link discovery without losing recall. In: 14th ACM SIGMOD Workshop on the Web and Databases (2011)
10. Li, J., Tang, J., Li, Y., Luo, Q.: RiMOM: a dynamic multistrategy ontology alignment framework. IEEE Trans. Knowl. Data Eng. **21**(8), 1218–1232 (2009)
11. Ngomo, A.C.N., Lehmann, J., Auer, S., Höffner, K.: RAVEN - active learning of link specifications. In: 6th International Semantic Web Conference Workshop on Ontology Matching, pp. 25–36 (2011)
12. Ngomo, A.C.N., Auer, S.: LIMES: a time-efficient approach for large-scale link discovery on the web of data. In: 22nd International Joint Conference on Artificial Intelligence, pp. 2312–2317 (2011)
13. Ngomo, A.C.N., Lyko, K.: EAGLE: efficient active learning of link specifications using genetic programming. In: Simperl, E., Cimiano, P., Polleres, A., Corcho, O., Presutti, V. (eds.) ESWC 2012. LNCS, vol. 7295, pp. 149–163. Springer, Heidelberg (2012)
14. Ngomo, A.C.N., Lyko, K.: Unsupervised learning of link specifications: deterministic vs. non-deterministic. In: 8th International Sematic Web Conference Workshop on Ontology Matching, pp. 25–36 (2013)
15. Nguyen, K., Ichise, R.: A heuristic approach for configuration learning of supervised instance matching. In: 14th International Semantic Web Conference Posters and Demonstrations Track (2015)
16. Nguyen, K., Ichise, R.: ScSLINT: time and memory efficient interlinking framework for linked data. In: 14th International Semantic Web Conference Posters and Demonstrations Track (2015)
17. Nguyen, K., Ichise, R., Le, B.: Interlinking linked data sources using a domain-independent system. In: Takeda, H., Qu, Y., Mizoguchi, R., Kitamura, Y. (eds.) JIST 2012. LNCS, vol. 7774, pp. 113–128. Springer, Heidelberg (2013)
18. Nikolov, A., d'Aquin, M., Motta, E.: Unsupervised learning of link discovery configuration. In: Simperl, E., Cimiano, P., Polleres, A., Corcho, O., Presutti, V. (eds.) ESWC 2012. LNCS, vol. 7295, pp. 119–133. Springer, Heidelberg (2012)
19. Niu, X., Rong, S., Zhang, Y., Wang, H.: Zhishi.Links results for OAEI 2011. In: 6th ISWC Workshop on Ontology Matching, pp. 220–227 (2011)
20. Papadakis, G., Ioannou, E., Palpanas, T., Niederée, C., Nejdl, W.: A blocking framework for entity resolution in highly heterogeneous information spaces. IEEE Trans. Knowl. Data Eng. **25**(12), 2665–2682 (2013)
21. Rong, S., Niu, X., Xiang, E.W., Wang, H., Yang, Q., Yu, Y.: A machine learning approach for instance matching based on similarity metrics. In: Cudré-Mauroux, P., et al. (eds.) ISWC 2012, Part I. LNCS, vol. 7649, pp. 460–475. Springer, Heidelberg (2012)

22. Song, D., Heflin, J.: Automatically generating data linkages using a domain-independent candidate selection approach. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 649–664. Springer, Heidelberg (2011)
23. Soru, T., Ngomo, A.C.N.: A comparison of supervised learning classifiers for link discovery. In: Proceedings of the 10th International Conference on Semantic Systems, pp. 41–44. ACM (2014)
24. Suchanek, F.M., Abiteboul, S., Senellart, P.: PARIS: probabilistic alignment of relations, instances, and schema. VLDB Endowment **5**(3), 157–168 (2011)
25. 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)