J Intell Inf Syst (2017) 48:519–551
DOI 10.1007/s10844-016-0426-3

ScLink: supervised instance matching system for heterogeneous repositories

Khai Nguyen[1,2] · **Ryutaro Ichise**[1,2]

Received: 28 October 2015 / Revised: 31 May 2016 / Accepted: 1 August 2016 /
Published online: 18 August 2016

Abstract Instance matching is the finding of co-referent instances that describe the same real-world object across two different repositories. For this problem, the heterogeneity, also known as the differences of objects' attributes and repositories' schema, is a challenging issue. It creates the limitations in the accuracy of existing solutions. In order to match the instances of heterogeneous repositories, a matching system can follow a configuration that specifies the equivalent properties, suitable similarity metrics, and other important parameters. This configuration can be created manually or automatically by learning methods. We present *ScLink*, an instance matching system that can generate a configuration automatically. In *ScLink*, we install two novel supervised learning algorithms, *cLearn* and *minBlock*. *cLearn* applies an apriori-like heuristic for finding the optimal combination of matching properties and similarity metrics. *minBlock* finds a blocking model, which aims at optimally reducing the pairwise alignments of instances between input repositories. In addition, *ScLink* introduces other techniques to take into account the scalability issue on large repositories. Experimental results on standard and very large datasets find that *minBlock* and *cLearn* are very effective and efficient. *cLearn* is also significantly better than existing configuration learning algorithms. It drastically boosts the accuracy of *ScLink* and makes the system outperform the state-of-the-arts, even when being trained using a small amount of labeled data.

Keywords Instance matching · Blocking · Schema-independent · Supervised · Configuration

✉ Khai Nguyen
nhkhai@nii.ac.jp

Ryutaro Ichise
ichise@nii.ac.jp

[1] National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan

[2] SOKENDAI (The Graduate University for Advanced Studies), Shonan Village, Hayama, Kanagawa 240-0193 Japan

1 Introduction

Instance matching solves the problem of detecting the different instances of the same object. Such instances are called co-referent instances. Instance matching is an important task in knowledge discovery and data mining, especially in data integration, data cleansing, and linked data interlinking (Hernández and Stolfo 1995; Rahm and Do 2000; Ferrara et al. 2011). It enables the data integration system to remain the consistency and integrity when combining different repositories. In linked data, instance matching is an essential component for linking the co-referent instances of different repositories. This process enriches the knowledge of linked data by connecting the independently created repositories. For its importance, instance matching has been extensively studied (Winkler 2006; Koudas et al. 2006; Köpcke and Rahm 2010). However, the achievement of an ultimate solution is still an open research problem.

In our work, we focus on instance matching for data in which the properties are declared. Some examples of this form of data are relational data, XML, and linked data. Our study is inspired by the problem of data integration in general and linked data interlinking, whose difficulties are almost shared but also varied in accordance with the characteristics of each problem. The first common difficulty is the heterogeneity coming from the schema and instance level. The heterogeneity of schemas arises from the fact that different repositories do not use the same schema, which contains the description of property (e.g., 'name' and 'range'). For example, 'abstract' can be named as 'comment', 'description', or even by a random code. The heterogeneity of instances results also from the inconsistent convention and the data collection from various sources. The same attribute of an instance can be represented in different styles (e.g., measurement unit, parts of name ordering, and time format) or even different values (e.g., New York and NY, clicker and remote control). The second common difficulty is the data ambiguity. Different values can describe the same information, and contradictorily, the same value can simultaneously refer to different things (e.g., Beverly Hills, California and Beverly Hills, Texas). The heterogeneity and ambiguity are clearly the causes of accuracy reduction.

In our work, although linked data is not the only one focus, it is the main motivation of scalability and heterogeneity. Those challenges on linked data are very difficult because of its open access and large scale. Most prominent linked data repositories are constructed from crowdsourced data, which offers a large opportunity to increase the knowledge. However, it also makes the data be vulnerable to the inclusion of incorrect or inconsistent information. Besides that, the heterogeneity and ambiguity of linked data repositories are much higher than traditional databases. For example, consider the DBpedia 3.9, there are 45,858 unique properties, 5,314,053 of only non-redirected instances,[1] and more than 20,000 instances sharing the token 'John' or 'William'. Matching repositories involved with such kind of data is very challenging in term of accuracy and scalability.

In order to perform the instance matching task, the matching configuration is used implicitly or explicitly to specify the settings of instances comparison (Ngomo and Auer 2011; Volz et al. 2009). A typical configuration includes the equivalent property mappings, similarity metrics, and similarity aggregation. One or many similarity metrics can be applied

[1] A redirected instance does not contain any information other than a URI linking to another instance that actually contains descriptions.

to a property mapping in order to consider many aspects of the given values. The similarity aggregation combines the similarities of all property mappings into a final score, which is used to determine the co-reference possibility.

Besides configuration-based approach, supervised instance matching can also be considered as a classification problem (Köpcke et al. 2010; Soru and Ngomo 2014). Different from the configuration-based approach as we discussed above, this approach takes the prediction result as the co-reference possibility. The limitation of the classification-based approach is that the classifier may be complicated to be interpreted. Furthermore, the post-processing, which was proved to be effective (Nguyen et al. 2012a; Ngomo et al. 2011), requires the explicit matching scores which are not offered by a classifier.

Instance matching is a well-studied problem. The early state of instance matching is the manually operated systems (Cruz et al. 2009; Niu et al. 2011), in which the configuration is constructed by the human. The heterogeneity of repositories is a factor of automation limitation for this approach because it blocks the portability of the constructed configuration. In addition, since user's experience does not guarantee to cover every data domain, the useful property mappings, suitable similarity metrics, and the optimality of other settings are difficult to be precisely decided. Finally, the accuracy is reduced as a consequence. Matching configuration can be effectively and automatically constructed by using learning algorithms. Unsupervised learning is an option for a fully automatic system (Hall et al. 2008; Ngomo and Lyko 2013; Nikolov et al. 2012; Bhattacharya and Getoor 2006). However, it is still far from the practical usage because it delivers just a modest accuracy but suffers from very high complexity. Contradictorily, supervised learning of configuration has the advantages in high accuracy and low complexity. Although this approach requires some labeled instance pairs for the learning step, a small size of training data can compensate the loss of automation. Configuration learning has been the focus of a few systems (Nikolov et al. 2012; Ngomo and Lyko 2012; Isele and Bizer 2013). However, those systems are not suitable for highly heterogeneous and large-scale datasets because they need manual interventions or are of high complexity.

We present $ScLink$, a scalable and supervised instance matching system. $ScLink$ contains two phases: *learning phase* and *resolution phase*. In *learning phase*, $ScLink$ automatically generates property mappings and assigns similarity metrics to each mapping in order to create the initial similarity functions. An optimal combination of these similarity functions and also other parameters of the *resolution phase* are selected by a learning algorithm.

$ScLink$ contains three main contributions. The first one is $cLearn$, a heuristic-based configuration learning algorithm. The second one is $minBlock$, a learning algorithm for blocking model. Blocking is a technique used for quickly detecting the candidates of potentially co-referent instances. The objective of this step is to avoid taking the comparisons for all pairwise alignments of instances between input repositories. The third novelty is the modified BM25 ($mBM25$[2]), a novel similarity metric that is robust to the ambiguous strings. $ScLink$ is highly scalable because its simple architecture enables the parallel processing for the comparisons of instances. In addition, different from previous supervised

[2]This similarity may be first proposed in another paper, which is being under review. Different from that paper, in this article, we empirically analyze its effectiveness.

systems, $ScLink$ optimizes the blocking model and thus generates fewer candidates. That is, the complexity is much reduced.

$ScLink$ is an extensive improvement of $cLink$ (Nguyen and Ichise 2015a). Compared to $cLink$, the scalability, ambiguity, and heterogeneity are more radically solved in $ScLink$. The novelties of $ScLink$ compared to $cLink$ include the workflow, $minBlock$ algorithm, and the $mBM25$. With the inclusion of the new elements, the experiments are more elaborately conducted with more datasets. In addition, the $cLearn$ algorithm is deeper analyzed in this paper. In short, the main contributions of $ScLink$ are the $minBlock$, $cLearn$, and the $mBM25$.

We analyze the performance of $ScLink$ in many aspects. We use 15 real datasets whose sizes vary from small to very large. We evaluate in detail $minBlock$, $cLearn$, and $mBM25$. We re-implement recent configuration learning algorithms (Isele and Bizer 2012, 2013; Nikolov et al. 2012; Hu et al. 2014; Ngomo and Lyko 2012) for comparing with $cLearn$ in the context of using the same input of similarity functions and all other parameters. We compare $ScLink$ with the previous systems, including the supervised and the state-of-the-art systems. Interestingly, experimental results find that our system needs only a small amount of training data for constructing an effective configuration. It supports the applicability of $ScLink$ in practical instance matching problems.

The rest of the paper is organized as follows. Section 2 defines the problem of instance matching and provides some preliminary concepts. Section 3 describes $ScLink$ in detail. Section 4 reports the experiments. Section 5 summarizes the related work. Section 6 reviews our study and outlines the future work.

2 Preliminaries

The problem of instance matching is defined as follows. Given two repositories: the source R_S and the target R_T. Each repository is a collection of instances and is associated with a schema. In this paper, the schema of a repository is defined as all properties existing in that repository. In addition, the set of attributes of an instance x described by property p is formatted as $p(x)$. In relational databases, an element of $p(x)$ can be considered as a cell value. In linked data, where the RDF statement $\langle subject, predicate, object\rangle$ is used to describe a fact, the representation of $p(x)$ is equivalent to $predicate(subject)$, which returns all related $objects$. Due to the heterogeneity, $p(x)$ may be empty (e.g., missing value), contains one, or multiple elements (e.g., different names for one thing).

The objective of instance matching is to identify the co-referent set $I \in R_S \times R_T$. A pair (x, y) of instances $x \in R_S$ and $y \in R_T$ belongs to I if x and y co-describe the same object. In the supervised scenario, a subset of $R_S \times R_T$ is labeled and given to the learning algorithms. When the learning process finishes, the obtained knowledge is applied for detecting the co-references existing in the unlabeled set.

The discrimination of source and target repository enables the role assignment for the input repositories. Most researchers consider the target as a referent repository, which is expected to contain the instances that are co-referent with many instances of the source repository. Therefore, the target repository contains more instances and so that it is more heterogeneous and ambiguous. This discrimination is helpful for reducing the complexity and improving the accuracy of the instance matching. These advantages will be further discussed in the next section.

The evaluation of instance matching is based on three measures: $recall$, $precision$, and $F1$ score. Among them, $F1$, which is the harmonic mean of $recall$ and $precision$ is more

Table 1 A motivating example

Source repository R_S			Target repository R_T			
	Name	Class		FirstName	Alias	Type
x_1	Alice Brady	Red	y_1	Alice	Alice	
x_2	Jack Eve	Pink	y_2	Eve	Snow	Pink
x_3	Jack Sloss	Pink	y_3	Jack	Alice	Black
x_4	Lee, Jack	Red	y_4	J.	Lee	Red
x_5	Bob Dylan					
$R_{SL} = \{x_1, x_2, x_3, x_4\}$, $R_{SU} = \{x_5\}$						
Co-references: (x_1, y_1), (x_2, y_2), (x_4, y_4)						

prioritized because it reflects the conciliation of *recall* and *precision*. Considering I as the expected co-references and D as the result of the instance matching process, the calculations of *recall* and *precision* are as follows.

$$recall(I, D) = \frac{|I \cap D|}{|I|} \tag{1}$$

$$precision(I, D) = \frac{|I \cap D|}{|D|} \tag{2}$$

We use an example of two small repositories to illustrate the detail of *ScLink*. Table 1 depicts this example. In the next section, we describe the detail of *ScLink* and use this example as an explanation for the main idea. In addition, for efficient reference, Table 2 lists the notations frequently used in this article.

3 *ScLink*

We begin with the description of the general workflow and then detail the steps.

3.1 Overview

The general workflow of *ScLink* is depicted in Fig. 1. The instance matching process consists of two phases: learning and resolution. The *learning phase* contains four steps: *property*

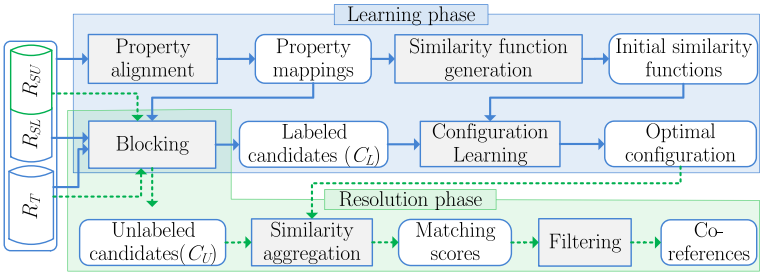

Fig. 1 The general architecture of *ScLink*

Table 2 List of frequently used notations

Notation	Meaning	Notation	Meaning
R	repository	C_L	labeled candidates
R_S	source repository	C_U	unlabeled candidates
R_T	target repository	R_{SL}	part of R_S for generating C_L
$schema(R)$	schema of repository R	R_{SU}	part of R_S for generating C_U
x, y, z, t	instance	B_{def}	default blocking model
p	property	B_{opt}	optimal blocking model
$v = p(x)$	attribute value of x at property p	sim	similarity function
w	string token	E	preprocessing function

alignment, *similarity function generation*, *blocking*, and *configuration learning*. The *resolution phase* contains four steps: *blocking*, *similarity aggregation*, and *filtering*. $ScLink$ takes two repositories R_S and R_T as the input. R_S is also separated into two parts: R_{SL} and R_{SU}.

In *learning phase*, first, the *property alignment* generates the property mappings using an overlap measure over the values described by all instances of R_S and R_T. According to the type of each mapping, the *similarity function generation* assigns the similarity metrics and produces the initial similarity functions. Each function computes one atomic score reflecting an aspect of the similarities between two instances. In parallel, in *blocking* step, $minBlock$ algorithm finds an optimal blocking model using the generated property mappings. The learned model is used to create the labeled candidates set C_L and unlabeled candidates C_U, where $C_L \in R_{SL} \times R_T$ and $C_U \in R_{SU} \times R_T$. In this step, an annotation process is conducted to create the training data for $minBlock$ as well as $cLearn$. Together with the initial similarity functions, the labeled candidates C_L are input into *configuration learning*, in which $cLearn$ is executed. $cLearn$ finds the optimal configuration, which contains the specification of further steps, including *similarity aggregation* and *filtering*.

In *resolution phase*, the optimal configuration is used to detect the co-references from unlabeled candidates. For each candidate, the *similarity aggregation* executes the learned similarity functions and combine the results into one final matching score. Finally, *filtering* step produces the co-references for unlabeled candidates by applying a constraint that considers the matching score of all candidates.

The main sub-problems of our work are the property alignment, $minBlock$, and $cLearn$. The objective of property alignment is finding the property mappings that are useful for matching instances, given the schema of the repositories. The $minBlock$ algorithm finds the effective blocking functions for generating fewer candidates but reserving high recall. $cLearn$ mainly searches for an optimal combination of similarity functions and similarity aggregator. All of them are at very high complexity because the goal is to find the optimal solution in a large collection of possible answers. The search space for those problems is at least $2^n - 1$, where n is the $|schema(R_S)| \times |schema(R_T)|$, the number of blocking functions, and similarity functions, with respective to above sub-problems. For $cLearn$, the algorithm also finds the optimal similarity aggregator. Hence, the search space is m times larger, where m is the number of possible aggregators.

Most parts of the above workflow are shared between many configuration-based supervised instance matching systems (Hu et al. 2014; Ngomo et al. 2011; Isele and Bizer 2012, 2013). Considering a complete system, one originality of $ScLink$ is that it also applies the learning for *blocking* step to create the optimal model, which can generate compact

candidate sets. Other systems use all property mappings for this step and thus, ignore the quality judgment for the input information because those mappings are not guaranteed to be correct. In other words, using all of them may produce many unnecessary candidates and add more limitations to scalability.

3.2 Property alignment

The mission of *property alignment* is to find the property mappings that are expected to describe the same attributes. For solving this, $ScLink$ first selects the property candidates from $schema(R_S)$ and then aligns them to the appropriate properties in $schema(R_T)$, where $schema(R_S)$ and $schema(R_T)$ are all properties of R_S and R_T, respectively.

3.2.1 Select property candidates from source repository

$ScLink$ does not take all properties of the source repository into comparison. Instead, only the properties that satisfy the requirements of diversity and frequency are considered. These measures are modified from the discriminability and coverage, which are designed in Song and Heflin (2011). The diversity (div) of a property p reflects how many unique values are described by p. Meanwhile, the frequency (frq) of p expresses how many instances contain p. These measures are calculated by (3) and (4), respectively.

$$div(p, R_S) = \frac{|\bigcup_{x \in R_S} p(x)|}{\sum_{x \in R_S} |p(x)|} \tag{3}$$

$$frq(p, R_S) = \frac{|\{x | x \in R_S, p(x) \neq \emptyset\}|}{|R_S|} \tag{4}$$

We separate the properties by their type before performing the selection. This mechanism enables the user to set a quota for each type of property, in order to increase the inclusion of useful properties from various types. We discriminate the properties into four types: *string*, *number*, *date*, and *URI*. For each type, $ScLink$ limits the search space into only properties whose diversity satisfies a threshold t_{div}. Then, a property is considered as a candidate if it is among the top K_{frq} most frequent properties.

3.2.2 Align properties between source and target repository

Each selected property from source repository is aligned with the properties having the same type of the target. Each alignment also is called a mapping. The confidence of a mapping $[p_S, p_T]$ is measured by counting the sharing values of p_S and p_T. Concretely, it is defined as follows:

$$\begin{aligned} conf([p_S, p_T]) &= \frac{|V_{p_S} \cap V_{p_T}|}{|V_{p_S}|} \\ V_{p_k} &= \bigcup_{x \in R_k} \bigcup_{v \in p_k(x)} E(v) \end{aligned} \tag{5}$$

where E is a preprocessing function. Table 3 is the detail of E. According to this table, E is designed for working with various data types. Among the mappings related to one property of the source repository, we select the mappings that belong to the top K_{conf} most confident ones. In technical aspect, a very small threshold is applied to the confidences to skip the slightly relevant mappings. We fix this value to 0.01.

Table 3 Value preprocessing

Data type	Returned values of E
string	unique set of tokens with stopword removal
number	rounded value with three decimal digits
date	original value
URI	original value

As an illustration, consider the example in Table 1, the $frq(\text{Name}, R_S)$ and $div(\text{Name}, R_S)$ are both 1.0 because it appears in all instances and all values are different. However, $frq(\text{Class}, R_S)$ and $div(\text{Class}, R_S)$ are $4/5$ and $2/2$, respectively. Suppose that $t_{div} = 0.5$ and $K_{frq} = 2$, then both Class and Name are selected for the first step. In the second step, the confidence of the mappings is listed in Table 4. Let $K_{conf} = 2$, all mappings whose confidence is higher than zero are selected as the final result.

K_{frq}, t_{div}, and K_{conf} can be chosen based on the number of retrieved property mappings. For each data type, the maximum quantity of mappings is $K_{frq} \times K_{conf}$ and additionally depends on t_{div}. More property mappings cause higher complexity for the later learning algorithms but may compensate more accurate result. In order words, setting these parameters is a heuristic for balancing the accuracy and complexity.

The diversity and frequency are computed only for properties of the source repository, instead of both repositories. This strategy originates from the assumption that the target is a reference repository (Section 2). The target property p_T is required to cover many values described by the source property p_S. Therefore, it is sufficient to measure the confidence directly without applying the property selection for the target repository.

ScLink generates the initial property mappings by observing the values described by the properties. That is, the creation of mappings is independent with the schema description. Therefore, *ScLink* is classified as a schema-independent system. In other words, the heterogeneity of schema is solved. However, this is just half of the solution because the quality of property mappings are not validated. The refinement is done by *configuration learning* step.

Some systems also applied similar techniques to find the property mappings (Nguyen et al. 2012a; Araujo et al. 2015). However, the prior property selection from source repository was not investigated although the computation of the confidence for all pairwise mappings between $schema(R_S)$ and $schema(R_T)$ is impractical on large datasets. In addition, other systems use Jaccard index to measure the confidence, by replacing the denominator of (5) by $|V_{p_S} \cup V_{p_T}|$. Quantifying such value is expensive because all elements of V_{p_T} must be retrieved. For large reference repository, querying V_{p_T} is not trivial. For example, assume that the property *label* exists in almost all instances, V_{label} are at very large size and the executions of preprocessing function E also increase consequently. Our strategy remains correct because we rank the mappings by each property of the source repository independently.

Table 4 Example: confidence of property mappings

	FirstName	Alias	Type
Name	3/7	2/7	0
Class	0	0	2/2

All property mappings are used for *similarity function generation* and the mappings of string properties are used in *blocking* step. In the next two sections, we describe the detail of those steps.

3.3 Similarity function generation

The task of this step is to assign the suitable similarity metrics to the property mappings. The result of one assignment is a similarity function. This step creates only the prototype of similarity functions. The execution of them is conducted by the *similarity aggregation* step, which is described in Section 3.6. We denote a similarity function as $sim^{metric}_{mapping}$. The formal definition of a similarity function is given in (6).

$$sim^{metric}_{[p_S,p_T]}(x, y) = \max_{v_x \in p_S(x), v_y \in p_T(y)} metric(v_x, v_y) \tag{6}$$

where max operator is used to return only the similarity of the most similar values described by p_S and p_T, because $p_S(x)$ or $p_T(y)$ may contain multiple facts. For example, given $name(x) = \{$'Sony', 'Sony Corporation'$\}$ and $label(y) = \{$'Sony Global'$\}$. Then, $sim^{Jaro}_{[name,label]}(x, y)$ compares the pairwise values of $name(x)$ and $label(y)$ using Jaro similarity and returns the maximum value.

ScLink assigns only the similarity metrics that are suitable for the type of the interested property mapping. We discriminate 4 types of mappings, including *string, number, date*, and *URI*. The list of similarity metrics for those types is reported in Table 5. According to this table, exact matching is used for comparing URI and $date$. The similarity of values of URI and $date$ is considered as 0 or 1 because it is suitable for the real data (e.g., homepage, birth date, and release date). Although the variation of time is useful in temporal instance matching (Christen and Gayler 2013), the focus of *ScLink* is the general context. For $number$, the inverse difference (7) is used to calculate how much close two numbers a and b are. Exact matching can also be used for this data type. However, inverse difference is more flexible when the repositories contain slightly different values (e.g., geographic coordinate).

$$invDiff(a, b) = \frac{1}{1 + |a - b|} \tag{7}$$

We apply Levenshtein metric for short strings comparison because it is effective for this kind of data (Levenshtein 1966). For long strings, we install the well-known TF-IDF Cosine and introduce the novel modified BM25 metrics. These metrics take the tokens (w) of the given strings into the computation, together with their normalized term frequency

Table 5 Similarity metrics by data type

Data type	Similarity metrics
string	Levenshtein (for short string), TF-IDF Cosine, modified BM25
number	inverse difference
date	exact matching
URI	exact matching

(TF) and the inverse document frequency (IDF). Equations (8) and (9) are the customized calculations of those weights, for adapting with the instance matching scenario.

$$TF(w, x, R) = \frac{\sum_{p \in schema(R)} |\mathcal{V}(w, p, x)|}{\max_{t \in R} \sum_{p \in schema(R)} |\mathcal{V}(w, p, t)|} \tag{8}$$

$$IDF(w, R) = \log \frac{|R| - \sum_{p \in schema(R)} |\{x | x \in R, \mathcal{V}(w, p, x) \neq \emptyset\}|}{\sum_{p \in schema(R)} |\{x | x \in R, \mathcal{V}(w, p, x) \neq \emptyset\}|} \tag{9}$$

where

$$\mathcal{V}(w, p, x) = \{v | v \in p(x), w \in E(v)\}$$

By including (8) and (9), the TF-IDF Cosine and the modified BM25 similarities of two token lists a and b belonging to $p_S(x)$ and $p_T(y)$ are calculated as in (8) and (9), respectively.

$$\begin{aligned} cosine(a, b) &= \frac{\sum_{w \in a \cap b} TFIDF(w, x, R_S) \times TFIDF(w, y, R_T)}{\sqrt{\sum_{w \in a} TFIDF(w, x, R_S) \times \sum_{w \in b} TFIDF(w, y, R_T)}} \\ TFIDF(w, x, R) &= TF(w, x, R) \times IDF(w, R) \end{aligned} \tag{10}$$

$$mBM25(a, b) = \frac{|a \cap b|}{|a \cup b|} \times \sum_{w \in a \cap b} tWeight(w, x, y) \times tEdit(w, a, b) \tag{11}$$

where

$$\begin{aligned} tWeight(w, x, y) &= IDF(w, R_S) \times IDF(w, R_T) \times \frac{TF(w, y, R_T) \times k_1}{TF(w, y, R_T) + k_2} \\ tEdit(w, a, b) &= invDiff(pos(w, a) - pos(w_0, a), pos(w, b) - pos(w_0, b)) \end{aligned}$$

The TF-IDF Cosine is popularly used for its advantages in weighting the tokens. However, it is sensitive to ambiguity because it ignores the tokens order, an important information. Using a similarity metric with a robust disambiguation capability is very important. We introduce the modified BM25 as a more effective metric, which also considers this useful aspect. The first factor, $\frac{|a \cap b|}{|a \cup b|}$, is the Jaccard coefficient of a and b. The second factor is the sum of the token weighting $tWeight$ and the order inverse difference $tEdit$. The function pos returns the position of a token in its parent string. The variable w_0 denotes the first shared token between a and b. The combination of Jaccard coefficient, $tWeight$, and $tEdit$ is effective because we can simultaneously penalize the less overlapped strings, include the token weight, and consider the token order. $tWeight$ modifies the original BM25 weighting scheme (Robertson et al. 1994), which is originally designed for Information Retrieval. $tWeight$ eliminates the target document length in BM25 and introducing the $IDF(w, R_S)$. k_1 and k_2 are fixed equivalently to the defaults of BM25, which are 2.2 and 0.3, respectively. In our experiment, the modified BM25 shows its considerable effectiveness against the TF-IDF Cosine on highly ambiguous datasets.

We illustrate two motivating example of $mBM25$. We simplify the examples by considering that the weight of all tokens is same and thus $tWeight$ is skip in the calculation. The first example is the necessity of $tEdit$. Suppose we need to compare 'Tokyo (Prefecture of Japan)'(1) to 'Tokyo Prefecture'(2), and 'Tokyo, Japan'(3). The task is to discriminate that (1) and (2) refer to the same entity (Tokyo Metropolis) but (1) and (3) do not because (3) refer to the special wards of Tokyo. If we use only Jaccard measure, it fails to discriminate the cases, because it gives 2/4 for the two pairs. However, $tEdit$ for these pairs in turn is $invDiff(0,0)_{Tokyo} + invDiff(1,1)_{Prefecture}$ and $invDiff(0,0)_{Tokyo} + invDiff(3,1)_{Japan}$. Because the former is higher, the task is successfully done. The second example is the necessity of Jaccard measure. We compare 'Tokyo University'(4) to 'University of Tokyo'(5) and 'The Tokyo Metropolitan University'(6). We have $tEdit(4,5) = 5/4$ and $tEdit(4,6) = 3/2$. Therefore, $tEdit$ fails to realize that (4) is closer to (5) than (6). However, because Jaccard$(4,5) = 2/3$ and Jaccard$(4,6) = 2/4$, the respective final results are $10/12$ and $6/8$, which satisfy the expectation.

The TF-IDF Cosine and modified BM25 metrics compare the token using exact matching. Since we focus on real data, which infrequently contains token distortion, exact matching is effective. In addition, $ScLink$ is implemented with a flexible mechanism that is ready for the injection of new similarity metrics when needed.

3.4 Blocking

The input of this step includes the string property mappings, R_{SL}, R_{SU} and R_T. In this step, the labeled candidate set C_L and unlabeled set C_U are generated. A candidate is defined as a pair of instances and is expected to be the actual co-reference. This step is very important because it reduces the number of pairwise alignments between R_S and R_T. Therefore, together with the retention of many actual co-references, another mission of this step is to generate the candidate sets with compact sizes.

For each property mapping, we define a blocking function, which receives the input of instances pairs C and returns the pairs that satisfy the blocking mechanism. The input, for example, can be $R_S \times R_T$ or just a subset of this Cartesian product. A blocking function is a combination of two elements: property mappings and blocking mechanism. Blocking mechanism can be a simple exact matching or the advanced soundex and edit-distance. In $ScLink$, the blocking mechanism is to qualify a pair if the instances share the first token in the values described by the properties of interest. Using multiple blocking functions increases the recall but has more possibility to generate incorrect candidates. Therefore, selecting an optimal set of blocking function is very important.

The procedure of *blocking* step is as follows. First, we create a default blocking model B_{def}, which is the set of all blocking functions generated from all input property mappings. The result of applying a blocking model is the union of the results of each member. Then, a candidates set C_{L0} is generated as the result of $B_{def}(R_{SL} \times R_T)$, where R_{SL} is a portion of R_S. C_{L0} is passed through an annotation process in which positive or negative label is assigned for each candidate. The annotation can be solved by the achievements of other studies. For instance, combining automatic matching algorithms and crowd-sourcing-based technology (Demartini et al. 2013; Vesdapunt et al. 2014). After that, $minBlock$ algorithm learns an optimal blocking model B_{opt} from C_{L0} and B_{def}. Finally, B_{opt} is used to create C_L and C_U, where $C_L = B_{opt}(C_{L0})$ and $C_U = B_{opt}(R_{SU} \times R_T)$.

An optimal blocking model generates compact candidate sets with ignoring minimal actual co-references. For the learning of such model, we propose $minBlock$. $minBlock$ considers all input blocking functions as baseline and tries to remain the actual co-references

that can be generated by the baseline. Meanwhile, it tries to reduce the unexpected candidates. The pseudo code of $minBlock$ is written in Algorithm 1. In this pseudo code, C^+_{L0} is the positive candidates and $recall$ is calculated by (1). The idea of $minBlock$ is simple. First, the variable $minRec$ is assigned with a value estimated from the recall of using all blocking function (B_{def}) (line 2). $minRec$ is the lower bound for the optimal blocking model. By default, t_{loss} is set to 1.0 for the zero loss expectation. After that, in each main iteration controlled by line 3, $minBlock$ removes the blocking function that generates the highest number of candidates but satisfies $minRec$. This selection is reflected by line 8. The algorithm stops when there is no blocking function for removal (line 12).

Algorithm 1 *minBlock*

Input: Training set C_{L0}, decimal parameter t_{loss}, default blocking model B_{def}
Output: Optimal blocking model B_{opt}

```
1  B_opt ← B_def
2  minRec ← recall(C+_L0, B_def(C_L0)) × t_loss
3  repeat
4      size ← |B_opt(C_L0)|
5      remove ← ∅
6      foreach f_block ∈ B_opt do
7          B = B_opt \ f_block
8          if recall(C+_L0, B(C_L0)) ≥ minRec and size > |B(C_L0)| then
9              remove ← f_block
10             size ← |B(C_L0)|
11     B_opt ← B_opt \ remove
12 until remove = ∅
13 return B_opt
```

Consider the example in Table 1, from the property alignment, we have 3 property mappings: [Name, FirstName], [Name, Alias], and [Class, Type]. Let f_1, f_2, and f_3 in turn are the blocking function defined for those mappings with the same blocking strategy, to match the first token. The co-references (x_1, y_1), (x_2, y_2), and (x_4, y_4) are given to $minBlock$ as C^+_{L0}. By applying f_1, f_2 and f_3 on $R_{SL} \times R_T$, we get the respective candidates: $\{(x_1, y_1), (x_2, y_3), (x_3, y_3)\}$, $\{(x_1, y_1), (x_1, y_3), (x_4, y_4)\}$, and $\{(x_1, y_4), (x_2, y_2), (x_3, y_2), (x_4, y_4)\}$. The result of applying the initial $B_{def} = \{f_1, f_2, f_3\}$ is the union of the above candidate sets and it covers all the expected co-references. Therefore, with $t_{loss} = 1.0$, the final blocking model is required to produce all the three co-references (i.e., $minRec = 1.0$). In the first iteration controlled by line 3, the removal of each function in B_{def} is considered. Without f_3, the candidate (x_2, y_2) is not produced, while without f_1 or f_2, all co-references are still retrieved. f_1 produces two unnecessary candidates $((x_2, y_3), (x_3, y_3))$ while f_2 produces only one $((x_1, y_3))$. Therefore, f_1 is removed and B_{opt} becomes $\{f_2, f_3\}$. In the next iteration, because the removal of either f_2 or f_3 violates $minRec$, the algorithm stops.

The complexity of $minBlock$ is acceptable. The worst case happens when the **if** statement in line 8 is always hold. In that case, the complexity of $minBlock$ is the quadratic $O\left(\frac{n(n+1)}{2}\right)$, where n is the size of B_{def}. This complexity indicates a moderate computational effort.

The learning of blocking model is also investigated by many studies (Bilenko et al. 2006; Dalvi et al. 2013; Kejriwal and Miranker 2013). Among them, most similar to $minBlock$ is optimal hashing (Dalvi et al. 2013), which also tries to reduce the size of the candidate set

while reserving the recall. However, the problem definition of this hashing model is very different from ours and therefore the solution is varied.

The definition of blocking function enables the modification of blocking mechanism. That is, one or multiple blocking mechanisms can be used for each property mapping. However, $ScLink$ currently uses a simple token-based strategy (Papadakis et al. 2013) as the only mechanism. The advantage of this strategy is to retain the highest number of the expected co-references, compared to that of many other systems (Nguyen et al. 2012a; Mendes and Jakob 2011; Volz et al. 2009; Isele et al. 2011), which use token weighting. Although weighting approach generates less number of candidates, it is accompanied by a considerable drop in recall. The token-based comparison is used in many blocking strategies (e.g., attribute clustering, n-grams, and suffix array). The most recent benchmark is reported in Papadakis et al. (2016).

3.5 Configuration learning

The labeled candidates set C_L is separated into two sets, the training set $Train$ and validation set Val. Using $Train$, Val, initial similarity functions I_{sim} (Section 3.3), and similarity aggregators I_{agg} (Section 3.6), this step learns the optimal configuration C_{opt} that is most suitable for the input repositories. A configuration specifies the combination of similarity functions F_{sim}, the similarity aggregator Agg, the parameters σ_{sim} associated with each similarity function sim, and the parameter δ of the *filtering* step (Section 3.7). The mission of this step is to automatically assign the optimal value to all elements of C_{opt}.

We use $cLearn$, a heuristic search method to optimize the combination of the similarity functions and the similarity aggregator. The pseudo code of $cLearn$ is given in Algorithm 2. In this pseudo code, we use dot ('.') notation to represent the member access operator. The detail of the functions and parameters used in Algorithm 2 is as follows.

- K_{top} is used to adjust the maximum number of selected similarity functions. A higher value of K_{top} offers more opportunity for the optimal configuration but also may increase the complexity because it may add more iterations controlled by line 12. In $cLearn$, we assign $K_{top} = 16$ by default. We use such a high value of K_{top} so that possibly $cLearn$ does not miss the optimal configuration. In addition, in our experiment, using $K_{top} = 16$ is sufficient to find a good configuration in an acceptable time on a desktop computer.
- $Init$ creates a configuration by assigning Agg, F_{sim}, and δ with given values.
- $Match$ produces the co-references from a collection of candidates given a configuration. This step calls the *similarity aggregator* (Section 3.6) and *filtering* (Section 3.7). The *similarity aggregator* computes the matching score of candidates and the *filtering* produces the co-references.
- $FindThreshold$ assigns a value to δ, a parameter of co-reference filter. This function first selects the top $|Train^+|$ candidates with the highest matching score, where $|Train^+|$ is the number of the actual co-references in $Train$. Then, it assigns the lowest score of the correctly detected co-reference to δ.
- $Evaluate$ computes the performance of instance matching by comparing the generated results with the labeled data. In this function, the $F1$ score is used as the default performance metric. It is the harmonic mean of $recall$ and $precision$, as calculated by (1) and (2).

Algorithm 2 *cLearn*

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

```
1  C_agg ← ∅
2  foreach A ∈ I_agg do
3      visited ← ∅
4      foreach sim ∈ I_sim do
5          c ← Init(Agg ← A, F_sim ← sim, δ ← 0)
6          links ← Match(c, Train)
7          c.δ ← FindThreshold(links, Train)
8          c.σ_sim ← c.δ
9          F1 ← Evaluate(links, Train)
10         visited ← visited ∪ {[c, F1]}
11     candidate ← TopHighestF1(visited, K_top)
12     while candidate ≠ ∅ do
13         next ← ∅
14         foreach g ∈ candidate do
15             foreach h ∈ candidate do
16                 c ← Init(Agg ← A, F_sim ← g.c.F_sim ∪ h.c.F_sim, δ ← 0)
17                 links ← Match(c, Train)
18                 c.δ ← FindThreshold(links, Train)
19                 F1 ← Evaluate(links, Train)
20                 visited ← visited ∪ {[c, F1]}
21                 if g.c.F_sim ≠ h.c.F_sim and F1 ≥ g.F1 and F1 ≥ h.F1 then
22                     next ← next ∪ {[c, F1]}
23         candidate ← next
24     F1 ← Evaluate(argmax_{v∈visited}(v.F1).c, Val)
25     C_agg ← C_agg ∪ {[c, F1]}
26 [C_opt, F1] ← argmax_{c∈C_agg}(c.F1)
27 return C_opt
```

cLearn begins with checking each similarity function (line 4 to 10) and select the best K_{top} performed ones (line 11). The selected functions are stored using *candidate* variable. After that, *cLearn* finds the optimal combination among the K_{top} functions using Apriori approach (Agrawal et al. 1994) (line 12 to 23). In the first iteration of line 12, *candidate* contains the groups of single similarity function. The size of those groups increases along with the iteration, as the result of combining different groups. Line 16 reflects such combination by taking the union of $g.c.F_{sim}$ and $h.c.F_{sim}$. The heuristic used in this algorithm is the direct enhancement assumption. According to line 21, the performance of using a combination must not be less than that of the combined groups. This heuristic is reasonable as a list of similarity functions that reduces the performance has little possibility of generating a further list with improvement. The qualified combinations are carried to the next iteration by updating *candidate* variable (line 23). The algorithm stops when a new combination of similarity functions is not made (*candidate* is empty).

The validation set Val is used to increase the generality of the final configuration C_{opt}. Each iteration controlled by line 2 finds an optimal configuration with one similarity aggregator A. In other words, there are $|I_{agg}|$ configurations in C_{agg}. For selecting the most optimal one from C_{agg}, instead of just picking the configuration having the best performance on $Train$, Val is recommended. In case the labeled data is too small to be separated into training and validation sets, a cross-validation is reasonable.

We illustrate the main idea of $cLearn$, which is to select the optimal combination of similarity functions (line 12 to 23). From the example given in Table 6, the result of property alignment, and $minBlock$, we get 3 property mappings [Name, FirstName], [Name, Alias], [Class, Type] and 6 candidates produced by $B_{opt} = \{f_2, f_3\}$. In order to simplify the illustration, we assign only Jaccard measure to the mappings and thus we get 3 respective similarity functions to the above mappings sim_1, sim_2 and sim_3. Considering the average as the similarity aggregator, the results of applying all combinations are listed in Table 6. We skip the stable filtering and set the matching threshold to 0.25. That is, if a value in Table 6 satisfies 0.25, the respective candidate becomes a detected coreference. Assume that at the first iteration of line 12 to 23 $candidate$ is $\{sim_1, sim_2, sim_3\}$. In this iteration, the combinations of two functions are considered. According to Table 6, only the combination of sim_1 and sim_3 violates the constraint of line 21 because the performance (F1) is lower than using sim_1. Therefore, $candidate$ is changed to $\{\{sim_1, sim_2\}, \{sim_2, sim_3\}\}$ and advanced to the next iteration. In the second iteration, the combination $\{sim_1, sim_2, sim_3\}$ is checked. Because it is not better than $\{sim_1, sim_2\}$, $candidate$ becomes empty and the algorithm stops. Finally, the optimal combination is $\{sim_1, sim_2\}$. In this example, all combinations are visited. In our experiment, the checked combinations are much smaller than all possible cases.

We analyze the complexity by the number of the configurations that $cLearn$ has to check. In total the complexity is $O(|G_{inp}| \times (|F_{def}| + f(while)))$, where $f(while)$ is the complexity of the **while** block from line 12 to line 23. The complexity of this block depends on K_{top}, and the probability of the **if** statement. Let p_i be the probability of the hold cases for the **if** statement in the i^{th} iteration of the **while** block. The complexity of the i^{th} iteration is $f(while^i) = p_{i-1} \times \binom{K_{top}}{i+1}$, where the iteration is counted from 1 and $p_0 = 1.0$. That is, the size of $candidate$ may increase for the first few iterations (if $K_{top} > 3$) and then decrease. In the worst case, if p is equal to 1.0 for all iterations, the result will be the union of K_{top} similarity functions. Otherwise, lower value of p makes the algorithm finish faster.

$cLearn$ works under the principle of the well-known Apriori algorithm, where each similarity function is considered as an item as in Apriori. $cLearn$ begins with the consideration of each single similarity function and then checks their combinations. Since the goals of Apriori and our learning task are different, we customize the acceptance mechanism when checking a new combination of similarity functions. While in Apriori algorithm, the condition is uniformly fixed for all cases, we adaptively change the criterion for each combination, which is the heuristic discussed above.

Table 6 Example: cLearn algorithm

	(x_1, y_1)	(x_1, y_3)	(x_1, y_4)	(x_2, y_2)	(x_3, y_2)	(x_4, y_4)	F1
sim_1	0.5	0	0	0.5	0	0	0.80
sim_2	0.5	0.5	0	0	0	0.5	0.67
sim_3	0	0	1	1	1	1	0.57
sim_1, sim_2	0.5	0.25	0	0.25	0	0.25	0.86
sim_1, sim_3	0.25	0	0.5	0.75	0.5	0.5	0.75
sim_2, sim_3	0.25	0.25	0.5	0.5	0.5	0.75	0.67
sim_1, sim_2, sim_3	0.33	0.17	0.33	0.5	0.33	0.5	0.75

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 of complexity (Nikolov et al. 2012; Ngomo and Lyko 2012; Isele and Bizer 2013) but this algorithm is still time-consuming because it is based on random convergence. In addition, the genetic algorithm has many free parameters. *cLearn* uses a reasonable heuristic to reduce the complexity and minimize the parameters.

Most studies consider the supervised instance matching problem as the discovery of property mappings, instead of identifying them by annotation. The reason is the correct mappings in semantic aspect are different from the useful mappings for instance matching. For example, air transportation companies frequently share their ICAO (i.e. unique abbreviation name) and stock symbol; or full name can be matched with first name and last name, instead of only the same amount of information. Besides taking this issue into account, *ScLink* also combines the similarity metric and the mappings for directly optimizing the similarity functions. The motivation of this mechanism is that each metric offers different advantages on different properties with the association of particular repositories.

3.6 Similarity aggregation

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

$$\begin{aligned} mScore_{F_{sim}}(x, y) &= \frac{1}{valid(U_{F_{sim}}(x, y))} \sum\nolimits_{v \in U_{F_{sim}}(x,y)} v^k \times weight(y) \\ U_{F_{sim}}(x, y) &= \{sim(x, y) | sim(x, y) \geq \sigma_{sim}, sim \in F_{sim}\} \end{aligned} \tag{12}$$

where $k \in \{1, 2\}$, $valid$ is a counting function, $weight$ is a function weighting the target instance y, and σ_{sim} is the parameter for each similarity function sim, which is determined automatically by *cLearn* (at line 8). k controls the transformation for each similarity v. When $k = 1$, $mScore$ function acts as a first order aggregation. When $k = 2$, we have a quadratic aggregation. There are 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 pair (x, y) having similarities $sim(x, y) < \sigma_{sim}$ while the former does not. For $weight$ function, *ScLink* also provides two options. For *non-weighting*, $weight(y)$ simply returns 1.0. For $weighting$, the function returns:

$$weight(y) = \log_{\max_{t \in R_T} size(t)} size(y) \tag{13}$$

where $size(y)$ counts the number of RDF triples existing in y. By using (13), we assume that the instances containing more 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. In addition, *ScLink* provides a $restriction$ mechanism to enable or disable σ_{sim}. When disabling, all σ_{sim} parameters are set to zero instead of the learned values.

In total, there are 16 combinations of $weight$, $valid$, k, and $restriction$. Consequently, there are 16 different aggregators supported by *ScLink*. All of them are used to initialize I_{agg} in *cLearn* and let the algorithm select the most optimal one.

3.7 Filtering

This step produces the final co-references. A candidate's co-referent possibility is not directly concluded from its matching score. Instead of that, the matching scores of all related candidates are considered. We reuse the principle of stable marriage problem (Gale and Shapley 1962), which was evaluated to be effective on instance matching (Nguyen et al. 2012a; Ngomo et al. 2011). A candidate (x, y) (where $x \in R_S$ and $y \in R_T$) is eventually co-referent if $mScore(x, y)$ satisfies the following conditional statement:

$$mScore(x, y) \geq \max\left(\max_{z \in R_S} mScore(z, y), \max_{t \in R_T} mScore(x, t)\right) \tag{14}$$

In this step, $ScLink$ also uses a cut-off threshold δ to eliminate the incorrect candidates but satisfying (14). δ is assigned automatically by the learning algorithm $cLearn$. Only candidates whose scores satisfy the above condition statement and threshold δ are selected for the final results.

The basic idea of the *filtering* step is to rank each candidate in the local set of candidates sharing its source or target instance. It guarantees that for very ambiguous repositories, a highly similar pair (x, y) is not necessary to be co-referent if there exist other pairs (x, z) with higher similarity.

Above we have described the details of $ScLink$. Next, we report some implementation notes of $ScLink$.

3.8 Implementation notes

Since the processings of all steps are parallelizable, it is feasible to apply paralleling techniques to enhance the speed of the system. Although many advanced frameworks are available and are used by other systems (Urbani et al. 2010; Volz et al. 2009; Ngomo and Auer 2011; Hogan et al. 2012), we currently implement $ScLink$ with the basic multi-threading technique, because we are also interested in memory efficiency and cost saving. Concretely, we combine multi-threading with each step of the system workflow. We experimented with very large datasets on a resource-limited computer in order to confirm the scalability of the system.

$ScLink$ is installed as part of ScSLINT (Nguyen and Ichise 2015b), an efficient framework for instance matching. In ScSLINT, input repositories are indexed using the well-known inverted index structured for pairs of $\langle p, p(x)\rangle$ and $\langle p(x), x\rangle$. Therefore, the *property alignment* and *blocking* are enhanced at the maximum processing speed. The complexity of indexing is very small and linear to the number of instances in the repository.

4 Experiment

In order to elaborately evaluate $ScLink$, we conduct 5 experiments:

- Experiment 1 evaluates the performance of *blocking* step.
- Experiment 2 evaluates the performance $cLearn$.
- Experiment 3 analyzes the similarity metrics and similarity aggregators.
- Experiment 4 reports the runtime of $ScLink$.
- Experiment 5 compares $ScLink$ with other systems.

All experiments are conducted on a desktop computer equipped with one Intel core i7 4770K CPU and 16GB memory. The results of these experiments are reported in Section 4.3 to 4.7. Next, we describe the datasets and the mutual settings of all experiments.

4.1 Datasets

We use in total 15 datasets collected from the instance matching problem on relational databases and linked data. The summary of the datasets is given in Table 7. In this table, $|R_k|$, P_k, and $fact_k$ are the number of instances, properties and total facts (i.e., the attributes) existing in the repository k ($k \in \{S, T\}$), respectively. $|I|$ is the number of actual coreferences. The scale of a dataset is measured by the pairwise comparisons between the repositories. Following that, the average scale is 64.6×10^9 and the largest is 0.4×10^{12}, which are sufficient to see the scalability of $ScLink$.

The first 3 datasets (D1-D3) cover the bibliography and e-commerce domains and are collected from the instance matching problem on relational databases. These datasets have a small number of instances as well as simple schemas. The next 5 datasets (D4-D8) belong to medicine and disease domains. Among them, the first 3 datasets have medium size while the other 2 are larger. The last 7 datasets (D9-15) belong to people (peo), location (loc), and organization (org) domains and have very large size. Particularly, the datasets related to Freebase are at a huge size. The largest one contains nearly 402×10^9 pairwise instances between the source and the target repositories. The schemas of most linked datasets are very heterogeneous. Considering DBpedia, 45,858 different properties are used to describe the instances. In Freebase, such quantity is even more than 50 times larger. For the last 9 datasets (D7-D15), together with many instances, the huge number of facts increase the ambiguity and complexity. Especially, the last 7 datasets (D9-D15) are realized as very challenging for their scalability, heterogeneity, and ambiguity. 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

Table 7 Summary of used datasets

ID	Name	$\|R_S\|$	$\|R_T\|$	P_S	P_T	$fact_S$	$fact_T$	$\|I\|$
D1	DBLP-ACM	2,616	2,294	4	4	10,464	9,162	2,224
D2	ABT-Buy	1,081	1,092	3	4	2,580	3,419	1,097
D3	Amazon-GoogleProducts	1,363	3,226	4	4	5,337	9,719	1,300
D4	Sider-Drugbank	2,670	19,689	10	118	96,269	507,495	1,142
D5	Sider-Diseasome	2,670	8,149	10	18	96,269	69,544	344
D6	Sider-DailyMed	2,670	10,002	10	27	96,269	131,064	3,225
D7	Sider-DBpedia	2,670	4,183,461	10	45,858	96,269	232,957,729	1,449
D8	Dailymed-DBpedia	10,002	4,183,461	27	45,858	131,064	232,957,729	2,454
D9	NYTimes.loc-Geonames	3,840	8,514,201	22	14	42,302	112,643,369	1,729
D10	NYTimes.loc-DBpedia	3,840	4,183,461	22	45,858	42,998	232,957,729	1,917
D11	NYTimes.org-DBpedia	6,045	4,183,461	22	45,858	54,404	232,957,729	1,922
D12	NYTimes.peo-DBpedia	9,958	4,183,461	22	45,858	103,341	232,957,729	4,964
D13	NYTimes.loc-Freebase	3,840	40,358,162	22	2,455,627	43,037	912,845,965	1,920
D14	NYTimes.org-Freebase	6,045	40,358,162	22	2,455,627	59,111	912,845,965	3,001
D15	NYTimes.peo-Freebase	9,958	40,358,162	22	2,455,627	103,496	912,845,965	4,979

inconsistencies between the provided ground-truth and the downloaded dump data, because of the difference in the release dates. Therefore, we have to manually exclude in total 130 (only 0.298 %) source instances which are related to such inconsistencies. In additional, the three repositories NYTimes, DBLP, and ACM do not contain co-referent instances within themselves, while the other repositories do.

The above datasets are selected because they are real datasets with the variety of domains and sizes. 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. Morever, many systems have been recently tested on those datasets (Soru and Ngomo 2014; Köpcke et al. 2010; Euzenat et al. 2011). That enables the comparisons between $ScLink$ and others systems.

Although the source repository of the tested datasets is much smaller than the target, $ScLink$ will also work for large source repository because the methods used in $ScLink$ is not limited to the small source repositories. Furthermore, because the target repository is considered as a reference, if the source repository is too huge, it is possible to reduce the scale by partitioning it into smaller parts sharing some characteristics (e.g., schema and class). Analyses on very large source repository and partition methods are beyond the scope of this paper.

4.2 Experimental settings

There are 3 parameters in $ScLink$: t_{div}, K_{fre}, K_{conf}. We are interested in using the same parameters for all datasets and test cases in order to evaluate the sensitivity of $ScLink$ to parameters. We find that the selection of these parameters is not difficult. We first fix the K_{fre}, K_{conf} into 4 for all property types. That is, at most 64 property mappings (4 types $\times$ 4 properties from R_S $\times$ 4 properties from R_T) will be generated. In order to see the ability to select good similarity functions, a considerable amount of property mappings should be given to $cLearn$. 64 property mappings are comparatively many because it is at least 2.4 times larger than the number of properties of each source repository. By fixing K_{fre} like above, we gradually reduce t_{div} from 1.0 until at least one string property is selected from each source repository. $t_{div} = 0.5$ satisfies this expectation and we use this value uniformly for all experiments. We observe that varying t_{div} in the range 0.2 to 0.7 does not change the generated mappings on all datasets excepts the last 7 datasets, which are related to NYTimes. For NYTimes repository, using t_{div} higher than 0.5 does not return any property of string type.

For training data separation, we follow the workflow of $ScLink$ (Fig. 1). We split the source repository into two parts R_{SL} and R_{SU}. Part R_{SL} is used to generate the labeled candidates C_L and R_{SU} is used to generate unlabeled candidates C_U. The ratio of R_{SL} and R_{SU} is varied between experiments (e.g., cross-validation or percentage split). After that, C_L is separated into two sets with the ratio 80 %:20 %. These set are used to generate two labeled candidates set, the training set $Train$ (from 80 %) and validation set Val (from 20 %), which are used by $cLearn$. In the rest of this section, we denote *x % labeled data* as *using x % instances of source repository* as R_{SL}.

The split of R_{SL} and R_{SU} is based on the ground-truth data, the set I of actual co-references. Each element of I is a pair of one source and one target instances. Given the expected split ratio r, we randomly separate I into two sets I_L and I_U with the ratio $\frac{|I_L|}{|I_L|+|I_U|} = r$. The source instances of I_L and I_U are put in R_{SL} and R_{SU}. Up to this point, $R_{SL} \cup R_{SU}$ may not be equal to R_S because the I is not necessary to cover all instances of R_S. Those uncovered instances are split with the same ratio r and additionally put to R_{SL}

and R_{SU}. Finally, we get $R_{SL} \cup R_{SU} \equiv R_S$ and maintenance the ratio r between R_{SL} and R_{SU} the ratio r for instances that exist (and thus also do not exist) in I. C_L and C_U are generated from R_{SL} and R_{SU}, respectively. Following this manner, it is simple to assign the label to C_L, C_U, and the detected co-references. The final accuracy (e.g., recall, precision, and F1) in our experiment is measured by comparing the detected co-references for R_{SU} with $|I_U|$.

An important note is that for experiments with percentage split, in order to reduce the random noise, we always repeat 10 times running for each dataset and report the average results.

4.3 Experiment 1: blocking

In this experiment, we compare the recall and number of candidates when using and not using $minBlock$. First, we report the results of not using this algorithm. That is, all property mappings and all instances of R_S are used for *blocking* step. In other words, we measure the number of candidates ($size$) and the recall (rec) for $C_0 = B_{def}(R_S \times R_T)$. Table 8 reports those results for all datasets. In this table, $size$ is the number of generated candidates and rec stands for recall. The high rec values in this table reflect the effectiveness of using token-based blocking, especially using only the first token. The most difficult dataset is $D3$, on which the recall is 0.865 and is particularly lower than other datasets. However, it is acceptable because the most recently equivalent result on this dataset is only 0.835 (Kejriwal and Miranker 2015), by using the advanced tri-gram attribute clustering (Ngomo et al. 2011). The number of generated candidates is very small compared to all possible pairwise instances between R_S and R_T. For all 15 datasets, the total pairwise instances are 0.97×10^{12} and the number of generated candidates is 1.38×10^9. That is, about 99.9 % unnecessary candidates are removed. However, compared to the actual co-references, the generated candidates are generally still at a much larger size.

The objective of $minBlock$ is to remain the recall and reduce the unexpected candidates. Therefore, we evaluate the effectiveness of $minBlock$ by taking the ratios of those values between after and before using $minBlock$. The procedure of this experiment is as follows.

- Given two splits R_{SL} and R_{SU}, we first generate the results of not using $minBlock$, $C_{L0} = B_{def}(R_{SL} \times R_T)$ and $C_{U0} = B_{def}(R_{SU} \times R_T)$. C_{L0} is used for $minBlock$ to generate the optimal blocking model B_{opt}.
- After that, we generate the result of using $minBlock$ by taking $C_L = B_{opt}(R_{SL} \times R_T)$ and $C_U = B_{opt}(R_{SU} \times R_T)$.

Table 8 Result of blocking using all property mappings

ID	$size$	rec	ID	$size$	rec	ID	$size$	rec
D1	342,837	0.9933	D6	5,013	0.9939	D11	61,702,166	0.9880
D2	61,756	0.9262	D7	482,605	0.9538	D12	46,942,099	0.9970
D3	70,550	0.8654	D8	1,034,653	0.9780	D13	222,686,571	0.9875
D4	5,771	0.9721	D9	32,161,659	0.9676	D14	357,377,464	0.9770
D5	4,258	0.9535	D10	38,201,823	0.9718	D15	620,076,154	0.9912

– Finally, we calculate the two ratios: size reduction rate $rSize = 1 - \frac{|C_L|+|C_U|}{|C_{L0}|+|C_{U0}|}$ (larger is better) and recall reduction rate $rRec = 1 - \frac{|R \cap C_U|}{|R \cap C_{U0}|}$ (smaller is better).

We measure the change of *size* for both labeled set (C_L) and unlabeled set (C_U) because they are used for $cLearn$ and *resolution phase*. However, we only measure the change of recall for unlabeled set because the recall of labeled set is identical to the original set C_{L0}, as t_{loss} is used as 1.0, the default value. Note that, when we divide R_S into different training split, the recall and the size of C_{L0} and C_{U0} are probabilistically similar to those of the parent set C_0 (reported in Table 8).

First, we use 5 folds cross-validation so that all instances are in turn used for training as well as testing. For this test, the recall reduction $rRec$ is equal to 0.0 for most datasets, except D7 and D12. For D7 and D12, only on 1 random fold (out of 5 folds), 1 expected candidate is ignored. However, the size reduction $rSize$ is varied between datasets. Therefore, we report the detail in Table 9. According to this table, the reduction of $size$ is low on small datasets but very high on large datasets, when unexpected candidates are frequently available. For D9 to D15, in total, compared to not using $minBlock$ a considerable number of 1.26×10^9 candidates are discarded. These results confirm the effectiveness and importance of $minBlock$, for enhancing the scalability of $ScLink$. By limiting the candidates, the complexity of $cLearn$ and *resolution phase* is much reduced.

4.3.1 Effect of size of training data

For a supervised system, the size of training data is very important. Therefore, we also evaluate $minBlock$ with different small amounts of training data. We vary the ratio labeled data from 1 % to 15 % and analyze the trend of $rRec$ and $rSize$.

Generally, on most datasets, $rRec$ reduces with the increase of labeled data. In other words, the more labeled data is given, the more optimal blocking model is learned. The *coefficient of variation* for $rRec$ values when changing the amount of labeled data is under 1.0 for all datasets except D15. Particularly, for D6 and D9, $rRec$ is always equal to zero whatever amount of labeled data is given. Consider the average of all datasets, $rRec$ quickly reduces from 0.0089 to 0.0035, when labeled data is increased from 1 % to 8 %. After this point, the change of $rRec$ is not considerable as only 0.001 unit is reduced for the range 8 % to 15 %. The slight variation of $rRec$ between different settings of annotation effort and the early saturated value show that $minBlock$ can learn the optimal blocking model by using a small amount of labeled data.

The reduction of $rRec$ is accompanied with the drop in $rSize$. However, the reduction of $rSize$ is not considerable and gradually slow down with the increase of labeled data. In

Table 9 Cross validation result with $minBlock$

ID	$size$	$rSize$	$rRec$	ID	$size$	$rSize$	$rRec$	ID	$size$	$rSize$	$rRec$
D1	341,446	0.0041	0	D6	5,013	0.0000	0	D11	4,280,108	0.9306	0
D2	61,756	0.0000	0.0010	D7	471,953	0.0221	0	D12	19,548,179	0.5836	0
D3	70,550	0.0000	0	D8	951,135	0.0807	0.0008	D13	9,386,067	0.9579	0
D4	5,362	0.0709	0	D9	3,713,019	0.8845	0	D14	19,353,423	0.9458	0
D5	4,227	0.0073	0	D10	2,006,056	0.9475	0	D15	61,029,275	0.9016	0

average of all datasets, $rSize$ drops from 0.53 to 0.43 when 1 % and 15 % labeled data is given. This value is almost not different with that of using 5 folds cross-validation, whose average $rSize$ is 0.422. In addition, the *coefficient of variation* of every dataset is under 1.0. For above facts, the reduction of $rSize$ is within an acceptable range even we try to reduce $rRec$ by giving more labeled data to $minBlock$.

4.4 Experiment 2: performance of learning algorithms

In order to evaluate the effectiveness of our proposed learning algorithm, we compare the result of $ScLink$ when using $cLearn$ and when replacing it by other algorithms. We compare with top rank selection ($naive$), information gain based selection ($gain$), and genetic algorithm ($genetic$). The mechanisms of these algorithms are as follow.

- $naive$ selects the K_{top} similarity functions that obtain highest $F1$.
- $gain$ re-implements the idea of ADL (Hu et al. 2014), which selects the most discriminative property mappings by independently measuring the information gain of each property.
- $genetic$ follows the idea of EAGLE (Ngomo and Lyko 2012), Knofuss (Nikolov et al. 2012), GenLink (Isele and Bizer 2012), and ActiveGenLink (Isele and Bizer 2013), which use genetic algorithm to learn the matching configuration. We use binary array representation for the combination of similarity functions. We choose the exponential ranking for fitness selection, 0.7 for single point crossover probability, 0.1 for single point mutation probability, and 50 for the population size. We limit the maximum iteration to 1000 and also use early stop mechanism, which terminates the algorithm when F1 is saturated.

In order to implement other algorithms, we replace the lines from 3 to 23 of Algorithm 2 with the new algorithms. In other words, the mechanism of determining σ_{sim}, δ, and Agg remains the same of all algorithms. The reimplementation of other algorithms offers elaborate comparisons. It enables using the same input of similarity functions and the mechanism of determining other settings. More important, other algorithms are installed in the systems that are not scalable enough and thus cannot work with large datasets like D7 to D15.

Table 10 reports the average F1 score on each dataset of the tested algorithms when using 5 folds cross-validation. According to this table, the proposed algorithm consistently outperforms $gain$ and $naive$. For $genetic$, $cLearn$ is better than this algorithm on 9 out of 15 datasets. Although the harmonic means of $cLearn$ and $genetic$ look closed, when considering each fold separately, so that there are 75 tests (i.e. for 15 datasets and 5 folds), paired t-test finds that the improvement made by $cLearn$ is significant ($p = 0.0223$). The similar results are also recorded for comparing $cLearn$ with $naive$ ($p<0.0001$) and $gain$ ($p<0.0001$).

$Naive$ and $gain$ are the fastest algorithms because they check each similarity function independently and do not consider the combinations. For $cLearn$, since we fixed the K_{top} into 16, in theory, the maximum number of configurations is 2^{16} for the worst case, if $cLearn$ exhaustively checks all combinations of similarity functions. However, in fact, with the effect of the heuristic, $cLearn$ stops after checking averagely 177 configurations. Meanwhile, $genetic$ needs to check 316 configurations in average. That is, the proposed heuristic reduces almost 44 % the configurations while remaining the high accuracy, compared to $genetic$. In summary, it is concludable that $cLearn$ is more reliable and faster than other tested algorithms.

Table 10 F1 scores of *ScLink* when using *cLearn* and other algorithms

	cLearn	*genetic*	*gain*	*naive*
D1	0.9626	**0.9656**	0.9585	0.9585
D2	**0.6918**	0.6824	0.6214	0.6241
D3	0.6102	**0.6133**	0.5619	0.5700
D4	**0.9486**	0.9376	0.9019	0.9096
D5	**0.8536**	0.8535	0.7470	0.7342
D6	**0.8630**	0.7798	0.6710	0.6713
D7	0.6591	**0.6645**	0.4302	0.6397
D8	**0.7316**	0.7301	0.7236	0.7266
D9	**0.9121**	0.9115	0.8626	0.8596
D10	**0.9222**	0.9042	0.9175	0.9126
D11	0.9244	**0.9294**	0.8695	0.9169
D12	**0.9749**	0.9748	0.9365	0.9675
D13	0.9164	**0.9171**	0.8756	0.8741
D14	**0.9330**	**0.9330**	0.9020	0.9269
D15	0.9461	**0.9467**	0.9123	0.9159
H.mean	**0.8380**	0.8310	0.7541	0.7900

Bold number indicates the best result on the respective dataset

4.4.1 *Effect of size of training data*

Similar to Experiment 1, we also analyze the trend of F1 score when varying the size of labeled data. Small amounts of labeled data are given to *cLearn*, from 1 to 15 %. For each dataset, we conduct the random split 10 times and take the average result, which is depicted in Fig. 2. From this figure, the harmonic mean of F1 score quickly increases when labeled data is varied from 1 % (0.7272) to 4 % (0.8078). After that, it increases with a lower acceleration. At the setting of 13 %, *ScLink* reaches the value 0.8329, which is slightly better than using *genetic* with 80 % labeled data (Table 10). On most datasets, F1 score is

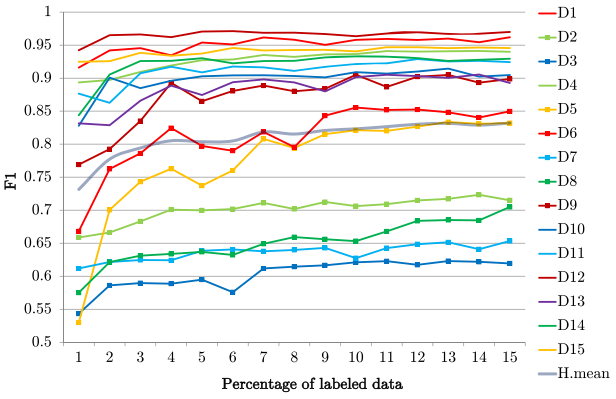

Fig. 2 Performance of *ScLink* with different amount of labeled data

near saturated at 5 % labeled data, except for D3, D5, D6, and D8. For D8, since F1 clearly increases at 15 %, we extend the experiment for this dataset and observe that the change slows down after 19 %. As 5 % training data is the goal of almost supervised systems, $ScLink$ expresses its capability by satisfying this expectation for most tested datasets.

4.5 Experiment 3: similarity aggregators and similarity metrics

In order to know which similarity aggregators and similarity metrics are effective, as well as the diversity of learned configurations, we conduct some statistics on the configurations produced by $cLearn$. We reuse the results of cross-validation in Experiment 2 for this analysis. As a result, there are 300 similarity aggregators and 2,419 similarity functions.

For similarity aggregator, considering each individual setting independently (e.g., $k = 1$ vs. $k = 2$ and $weighting$ vs. $non\text{-}weighting$), the following settings are most frequently selected: $k = 2$ (64 %), enabled $restriction$ (79 %), $valid = |U_{F_{sim}}(x, y)|$ (54 %), and $weighting$ (77 %). Among them, $weighting$ is very effective on the large datasets from D9 to D15 as it is always selected for all tests related to these datasets. All settings except $valid$ show the dominant proportions in the learned configurations. However, the dominance is not strong enough to confirm a universal effectiveness of each setting. For example, 21 % of cases requires another setting for $restriction$. The combination of above settings is also the most frequent aggregator. However, its share is only 24 % out of the 300 ones. This result implies that it is difficult to manually connect the dots between the datasets and the optimal similarity aggregator.

For similarity metrics, all string-related metrics show a relative balance as the similar proportions of Levenshtein (25 %), TFIDF-cosine (32 %), and $mBM25$ (28 %). In addition, $rDiff$ is very important for the subsets related to location domain as it is always selected on D9, D10, and D13, for the distance estimation of geographic coordinates.

An interesting finding is observed on D10. When the size of training data is 80 %, only *longitude*, one of two important geographic properties, is selected. While both *longitude* and *latitude* are considered as important for D9, D13, and even in human thinking, the learning algorithm returns a different recommendation. This example, together with the variety of similarity aggregators and similarity metrics as reported above, shows that for a particular input, it is difficult for a user to define a perfect matching configuration like an automatic system can do.

For the evaluation of $mBM25$, we compare the result of $ScLink$ when *using* and *not using* this metric. 5 folds cross-validation is also used for this test. The result is reported in Table 11. According to this table, by including $mBM25$, $ScSLINT$ improves the results for most datasets. Especially, for datasets from D10 to D14, which are among the large and

Table 11 F1 scores of *using* (use) and *not using* (not use) $mBM25$ in $ScLink$

ID	use	not use	ID	use	not use	ID	use	not use	ID	use	not use
D1	**0.9626**	0.9553	D5	0.8536	0.8536	D9	0.9121	**0.9179**	D13	**0.9164**	0.8819
D2	**0.6918**	0.6729	D6	0.8630	**0.8733**	D10	**0.9222**	0.8835	D14	**0.9330**	0.9087
D3	**0.6102**	0.5720	D7	**0.6591**	0.6506	D11	**0.9244**	0.9000	D15	0.9461	**0.9468**
D4	**0.9486**	0.9283	D8	0.7316	**0.7341**	D12	**0.9749**	0.9651	H.mean	**0.8380**	0.8227

Bold number indicates the best result on the respective dataset

highly ambiguous datasets, *using* $mBM25$ is clearly better. For D6, D8, D9, and D15, *using* $mBM25$ drops the performance but very slightly. For these datasets, the token order is not important because the learned similarity functions are almost constructed for short string properties and the strings that are described by such properties frequently contain only one token. In summary, $mBM25$ shows its effectiveness against TF-IDF Cosine, especially for large and ambiguous datasets.

4.6 Experiment 4: runtime

Efficiency is an important factor of instance matching systems. Therefore, we evaluate the runtime of $ScLink$ on tested datasets. Five percent labeled data is used for this experiment. For small datasets from D1 to D6, $ScLink$ takes only 2.68 seconds in average to complete all the steps. Especially for D2 and D5, the runtime is under 1.0 second. The longest runtime is also small, 8.1 seconds, measured on D1. For larger datasets, the runtime ranges from 38.6 seconds (D7) to 9.28 minutes (D15) and the average is 3.34 minutes. We are interested in comparing the runtime of $ScLink$ and other systems. However, the runtime of other systems on medium to large datasets (D7 to D15) are unfortunately not available. However, the speed of $ScLink$ is truly impressive as it is fast on D9 to D15, which are the scalability barriers for most existing systems.

In order to see the proportion of each step, we plot the detailed runtime of the medium and large datasets in Fig. 3. According to this figure, the consumed time of *property alignment* depends on target repository. In overall, for medium datasets like D7, D8, and D9, this step occupies the largest portion in the total runtime. For larger datasets, $cLearn$ and *similarity aggregation* share the dominant parts. In average, from D9 to D15, these steps cost 37.2 % and 41.2 %, respectively. Meanwhile, $minBlock$ algorithm is very efficient. It takes only 9.5 % of the runtime in average but delivers very important benefit. Without using $minBlock$, about 11.5 times longer runtime would be required for *similarity aggregation* step, as the reduction rate of candidates is about 91.3 % (Experiment 1). For example, compare to the base framework $ScSLINT$ (Nguyen and Ichise 2015b), the total runtime for D15 is 36 minutes because the number of candidates is more than 10 times larger (Nguyen

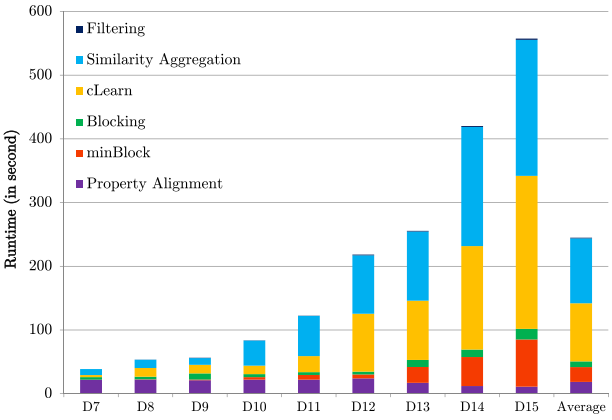

Fig. 3 Detailed runtime of $ScLink$ with 5 % labeled data

and Ichise 2015b). In summary, *ScLink* is very fast for large datasets and the consumed time of learning algorithms is also acceptable for the supervised scenario.

4.7 Experiment 5: comparison to other systems

We compare the performance of *ScLink* with a series of recent and state-of-the-art systems, including unsupervised, supervised, and non-learning-based ones. The availability of their result is also a criterion of selection. Since each system was tested on different datasets due to the support of data format and/or scalability, we separate the comparison into three groups: D1 to D3, D4 to D8, and D9 to D15.

4.7.1 D1 to D3

These datasets enable the comparison between *ScLink* and other instance matching system for relation databases. We compare *ScLink* with the state-of-the-art FEBRL (Christen 2008c) and MARLIN (Bilenko and Mooney 2003), and the most recent work in Kejriwal and Miranker (2015), which we temporarily call semiBoost. These systems are all classifier-based. FEBRL and MARLIN use SVM to learn the classification model. semiBoost uses AdaBoost in conjunction with semi-supervised learning.

The reported results of these systems are for different settings of labeled data. For semiBoost, 2 % is used. For FEBRL and MARLIN, 22 %, 46 %, and 38 % are used for D1, D2, and D3, respectively. Therefore, for each comparison, we give the same amount of labeled data to *ScLink*. Table 12 reports the results of *ScLink* and these systems.

4.7.2 D4 to D8

For these datasets, we compare *ScLink* with ObjectCoref (Hu et al. 2011) and the work in Rong et al. (2012), which we temporarily call rdBoost. ObjectCoref is a semi-supervised system that learns discriminative property mappings. rdBoost uses AdaBoost to train a committee of random forest classifiers. In addition, we include the results of RiMOM (Li et al. 2009) and PARIS (Suchanek et al. 2011) as two state-of-the-arts among automatic systems. RiMOM combines multiple matching strategies in order to obtain the optimal resolution result. PARIS automatically generates co-references of instances, properties, values, and classes by combining similarity and probability propagation.

rdBoost uses 5 % candidates for training and ObjectCoref uses 2.3 %, 11.6 % and 1.2 % for D1, D2, and D3, respectively. Note that for ObjectCoref, only the results on D1, D2, and D3 are reported. Table 13 reports the comparison on these datasets.

Table 12 F1 scores of *ScLink* and other systems on D1 to D3

Training data	System	D1	D2	D3	H. mean
2 %	*ScLink*	**0.9395**	**0.6434**	**0.5718**	**0.6869**
	semiBoost	0.9342	0.3913	0.3627	0.4700
Variable	*ScLink*	0.968	**0.713**	**0.612**	**0.7371**
	FEBRL	**0.976**	**0.713**	0.601	0.7333
	MARLIN	0.974	0.708	0.599	0.7302

Bold number indicates the best result on the respective dataset

Table 13 F1 scores of *ScLink* and other systems on D4 to D8

Training data	System	D4	D5	D6	D7	D8	H.mean
5 %	*ScLink*	**0.927**	**0.824**	**0.776**	**0.802**	**0.6354**	**0.781**
	rdBoost	0.903	0.794	0.733	0.641	0.375	0.628
Variable	*ScLink*	**0.897**	**0.821**	**0.774**			**0.827**
	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	0.208

Bold number indicates the best result on the respective dataset

4.7.3 D9 to D15

Many systems have reported the experiments on these 7 datasets. Unfortunately, in some of those experiments, the repositories are simplified in various manners and thus the challenge of ambiguity and scalability are much reduced. For example, instead of inputting the whole target repository, only the class (location, organization, or people) related the domain of source repository (NYTimes) is used by ADL (Hu et al. 2014); or only instances residing in actual co-reference are used by ADL (Hu et al. 2014), Knofuss (Nikolov et al. 2012), ActiveGenLink (Isele and Bizer 2013), and SLINT+ (Nguyen et al. 2012a). As using such simple datasets, it is not difficult to obtain the perfect results using a simple method, as reported in by SLINT+ (Nguyen et al. 2012a).

We report here the comparisons with Zhishi.Links (Niu et al. 2011), AgreementMaker (Cruz et al. 2011), and SERIMI (Araujo et al. 2011), which are tested on full datasets though they are not learning-based systems. In addition, because the periods of data retrieval are different, the repositories used for *ScLink* and other systems may be slightly different. For all those reasons, the comparisons are made for reference purpose. In order to minimize the difference of input knowledge, we feed only 5 % labeled data into the learning algorithms. Five percent is also used popularly as an expectation for the size of labeled data. Table 14 reports this comparison. The results of other systems are collected from Euzenat et al. (2011).

Table 14 F1 scores of *ScLink* and other systems on D9 to D15

	ScLink	Zhishi.Links	AgreementMaker	SERIMI
D9	0.87	**0.91**	0.85	0.80
D10	0.90	**0.92**	0.69	0.68
D11	**0.91**	**0.91**	0.74	0.88
D12	**0.97**	**0.97**	0.88	0.94
D13	0.88	**0.88**	0.85	**0.91**
D14	**0.93**	0.87	0.80	0.91
D15	0.94	0.93	**0.96**	0.92
H.mean	**0.913**	0.912	0.816	0.853

Bold number indicates the best result on the respective dataset

4.7.4 Discussion

In overall, *ScLink* expresses an impressive performance as it is better than other systems in many cases. Compared to state-of-the-art supervised systems on relational databases, *ScLink* is competitive to FEBRL and MARLIN. Compared to recent resolution systems for linked data, *ScLink* is far better than semiBoost, rdBoost, and ObjectCoref. Especially for D2 and D3, D7 and D8, which seem to be difficult for semiBoost and rdBoost because of the presences of co-references inside the source or target repository. *ScLink* clearly outperforms AgreementMaker and SERIMI, and is competitive to Zhishi.Links. Note that Zhishi.Links is specially customized for these datasets as this system applies 19 unification rules for matching difficult strings that frequently appear in this dataset (e.g., Co and Company, Manhattan and NYC). Therefore, considering the importance of generality, as well as the better results of *ScLink* in term of overall harmonic mean, *ScLink* reveals its strengths against Zhishi.Links. Comparing learning-based and non-learning-based systems, *ScLink*, rdboost, and ObjectCoref are much better than RiMOM and PARIS. This fact confirms the necessity of learning-based systems for improving the effectiveness.

5 Related work

Instance matching is an extensively studied problem. The literature of instance matching includes candidate generation methods, matching frameworks, and full solutions ranging from automatic to learning-based systems. In this section, we briefly summarize some representatives of each group.

Since candidate generation is a crucial step of most matching systems, the research on this sub-problem is attractive and is considered as an independent topic. Considering the complexity, token-based blocking (Papadakis et al. 2013) is the most basic approach as it is the fastest one. The token-based approach is also the most effective in term of recall. In most real cases, generating candidate by a comparison on the tokens is acceptable although in a few cases (e.g., availability of spelling errors), character-based is safer. The representatives of more advanced methods include sorted neighborhood (Hernández and Stolfo 1995), canopy clustering (McCallum et al. 2000), collective blocking (Isele et al. 2011; Nguyen et al. 2012a), and learning-based algorithms (Dalvi et al. 2013; Kejriwal and Miranker 2013; Papadakis et al. 2014). Sorted neighborhood arranges the instances into an order of lexicographic feature estimated from the value of instances. After that, the instances closing to the others within a fixed-size window are grouped into one block. Canopy clustering randomly selects a center and the instances that have an acceptable distance to the center is put into its cluster. Collective blocking takes multiple types of property mappings for generating candidates. Because this approach uses a multitude of information, it delivers high accuracy but also comes with it high complexity. Learning-based algorithms are can generate the most optimal candidate set. However, existing algorithms are complex and not suitable for large repositories. An important technique in blocking is to eliminate the instance groups with very large size (Papadakis et al. 2011). These cases frequently contain instances sharing the stopwords. In *ScLink*, the preprocessing step manages the solution for this issue (Table 3).

MOMA (Thor and Rahm 2007), FEBRL (Christen 2008c), SILK (Volz et al. 2009), and LIMES (Ngomo and Auer 2011) are among the famous instance matching frameworks. These frameworks allow the user to manually input the matching configuration. Among them, SILK and LIMES support matching of linked data repositories and are considered

as scalable frameworks, according to the reported experiments. SILK and LIMES also provide the integration of map-reduce technology. SILK and LIMES are the base framework for many recent instance matching algorithms. Most recently, ScSLINT is presented as a memory and time efficient framework, which is significantly faster than LIMES (Nguyen and Ichise 2015b). *ScLink* is also implemented as a part of this framework. Also focus on efficient instance matching, Kirsten et al. parallelized the instance comparisons on different blocks, together with memory caching technique (Kirsten et al. 2010). Similar to above framework, AgreementMaker (Cruz et al. 2009, 2011) and Zhishi.Links (Niu et al. 2011) are manual instance matching systems. These systems offer the mechanism to include domain knowledge, which is helpful for improving the accuracy.

Remarkable automatic systems include SERIMI (Araujo et al. 2015), SLINT+ (Nguyen et al. 2012a), KD2R (Pernelle et al. 2013), RiMOM(Li et al. 2009), and PARIS (Suchanek et al. 2011). SERIMI and SLINT+ automatically detect the property mappings using pure statistical methods and apply these mappings for comparing instances. The limitation of this approach is that mappings contain incorrect alignments and reduce the final accuracy. The scalable architecture of SLINT+ enables the introduction of its efficient version. *ScLink* is implemented on that scalable version of SLINT+. KD2R uses the nearly functional properties for matching instances. Such property is effective in discriminating ambiguous instances. However, this method is limited to only repositories having the same schema. RiMOM and PARIS are among the state-of-the-art systems although their mechanisms are totally different. RiMOM dynamically combines multiple comparison strategies in matching instances. PARIS simultaneously matches the instances, classes, properties, and values using probability propagation. Also based on probabilistic theory, but ZenCrowd (Demartini et al. 2013) is proposed as a crowd-sourcing based system. ZenCrowd ranks the candidates in each block and asks humans for the low ranked ones. In addition, collective instance matching (Dong et al. 2005; Bhattacharya and Getoor 2004; Locoro et al. 2014) is well-studied in graph-based data, such as the collaboration domain of relational database or linked data generally. The basic idea is that the similarity of one node affects those of the others using a propagation mechanism. PARIS can be classified into this approach as well.

Recent achievements of instance matching also include incremental and progressive instance matching. Whang et al. focused on the evolution of data and the matching rules (Whang and Garcia-Molina 2014). Stating that re-running the matching process when the input change is costly, the authors proposed algorithms to such incremental matching setting. In parallel independent work, Altowim et al. propose a progressive algorithm to rank the block by the predictive number of co-references (Altowim et al. 2014). The blocks are then processed in a programmatic order instead of random as in previous matching settings. This approach is useful if a system has to return an expected number of co-references within a limited time.

Most closed to our work are RAVEN (Ngomo et al. 2011), EAGLE (Ngomo and Lyko 2012), ActiveGenLink (Isele and Bizer 2013; Nikolov et al. 2012), and EUCLID (Ngomo and Lyko 2013). These systems also focus on learning of matching configuration. EAGLE, ActiveGenLink, and Knofuss apply the genetic algorithm to search for an optimal configuration while RAVEN and EUCLID use deterministic methods. RAVEN and ActiveGenLink also apply active learning for reducing the number of training examples. In addition, by focusing on a very similar problem, ADL (Hu et al. 2014) learns for discriminative property mappings that are expected to best separate the co-referent and non-co-referent cases. The limitation of ADL is that this system considers each mapping independently, and therefore cannot evaluate the quality of their combination.

Machine learning methods are widely studied as a solution for instance matching, by considering the matching of two instances as a classification problem. Many machine learning algorithms have been used, including CRF (Hall et al. 2008), latent Dirichlet (Bhattacharya and Getoor 2006), Neural network (Li and Clifton 2000), SVM (Christen 2008a), rule-based (Sheila et al. 2002), and decision tree (Nguyen et al. 2012b; Rong et al. 2012). A recent comparison between machine learning classifiers is reported in (Soru and Ngomo 2014). MARLIN (Bilenko and Mooney 2003) and the work in (Sarawagi and Bhamidipaty 2002) investigate committee-based active learning methods for training example selection. Together with active learning, transfer learning (Rong et al. 2012) and semi-supervised learning (Kejriwal and Miranker 2015; Hu et al. 2011) are alternative options for the compensation of the automation loss in supervised approach. Most machine learning algorithms try to maximize the margin between co-referent and non-co-referent cases. However, since most of them are probability-based, a conjunction of the classifier and post-processing (e.g. the *filtering* step of *ScLink*) is difficult, as well as classifier's interpretation, which is helpful for scaling up the *resolution phase*.

6 Conclusion

In this paper, we presented a supervised instance matching system named *ScLink*. We describe our solutions for the issue of heterogeneity, ambiguity, and scalability. We install in *ScLink* a novel learning algorithms for configuration and blocking model. We also use a robust string similarity metric. We reported the detail analyses and experiments for evaluating our proposed system. The experimental results confirm that *ScLink* meets practical demands in instance matching for real and large data. *ScLink* consistently outperforms other systems of the same objectives, including the state-of-the-arts. The heuristic search algorithm used in *ScLink* is also significantly better than many advanced methods of configuration learning. Together with effectiveness, *ScLink* is also more efficient in both time and memory consumption. The source code of *ScLink* is available at the project page: http://ri-www.nii.ac.jp/ScSLINT/.

A common shortage of all supervised systems is the availability of training data. Active learning is a promising strategy to reduce the annotation effort. In order to apply active learning, a study on effective training example selection is necessary because querying informative examples from a large pool of candidates is not a trivial problem. Although active learning has been studied by many researchers, such complexity issue was not convincingly solved (Christen 2008b; Sarawagi and Bhamidipaty 2002; Isele and Bizer 2013). Another alternative is transfer learning. As many existing co-references are retrievable nowadays, especially on the Web of linked data, the configuration constructed for two particular repositories can be shared to resolve the co-references between similar repositories (e.g., domain, schema, provider). Transfer learning is preliminarily evaluated that to work with instance matching problem (Rong et al. 2012). However, the issue of identifying exactly the equivalent properties between the trained repositories and the unknown repositories having different schema remains unsolved.

For the improvement of *ScLink* particularly, besides the above possibilities, *ScLink* can benefit from technical enhancement of similarity metric in term of processing speed (Mishra et al. 2013; Soru and Ngomo 2013). *ScLink* is not sensitive to its parameters,

however, reducing the free parameters can perfectly satisfy the automation mission. In addition, an analysis on very large source repositories and partitioning methods may support the applicability of $ScLink$.

References

Agrawal, R., Srikant, R., et al. (1994). Fast algorithms for mining association rules. In *Proceedings of the 20th international conference on very large data bases*, (Vol. 1215 pp. 487–499).

Altowim, Y., Kalashnikov, D.V., & Mehrotra, S. (2014). Progressive approach to relational entity resolution. *Proceedings of the VLDB Endowment*, *7*, 999–1010.

Araujo, S., De Vries, A., & Schwabe, D. (2011). SERIMI Results for OAEI 2011. In *Proceedings of the 6th workshop on ontology matching* (pp. 212–219).

Araujo, S., Tran, D.T., de Vries, A., & Schwabe, D. (2015). SERIMI: Class-Based matching for instance matching across heterogeneous datasets. *IEEE Transactions on Knowledge and Data Engineering*, *27*(5), 1397–1440.

Bhattacharya, I., & Getoor, L. (2004). Iterative record linkage for cleaning and integration. In *Proceedings of the 9th SIGMOD workshop on research numbers in data mining and knowledge discovery* (pp. 11–18): ACM.

Bhattacharya, I., & Getoor, L. (2006). A latent dirichlet model for unsupervised entity resolution. In *Proceedings of the 6th SIAM international conference on data mining* (pp. 47–58): SIAM.

Bilenko, M., & Mooney, R.J. (2003). Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the SIGKDD conference on knowledge discovery and data mining* (pp. 39–48): ACM.

Bilenko, M., Kamath, B., & Mooney, R.J. (2006). Adaptive blocking: Learning to scale up record linkage. In *Proceedings of the 6th international conference on data mining* (pp. 87–96).

Christen, P. (2008a). Automatic record linkage using seeded nearest neighbour and support vector machine classification. In *Proceedings of the 14th SIGKDD international conference on knowledge discovery and data mining* (pp. 151–159): ACM.

Christen, P. (2008b). Automatic training example selection for scalable unsupervised record linkage. In *Proceedings of the 12th pacific-asia conference on advances in knowledge discovery and data mining* (pp. 511–518): Springer.

Christen, P. (2008c). Febrl: a freely available record linkage system with a graphical user interface. In *Proceedings of the 2nd australasian workshop on health data and knowledge management*, (Vol. 80 pp. 17–25).

Christen, P., & Gayler, R.W. (2013). Adaptive temporal entity resolution on dynamic databases. In *Proceedings of the 17th pacific-asia conference on advances in knowledge discovery and data mining* (pp. 558–569): Springer.

Cruz, I.F., Antonelli, F.P., & Stroe, C. (2009). AgreementMaker: Efficient matching for large real-world schemas and ontologies. In *Proceedings of the VLDB endowment*, (Vol. 2 pp. 1586–1589).

Cruz, I.F., Stroe, C., Caimi, F., Fabiani, A., Pesquita, C., Couto, F.M., & Palmonari, M. (2011). Using agreementMaker to align ontologies for OAEI 2011. In *Proceedings of the 6th workshop on ontology matching* (pp. 114–121).

Dalvi, N., Rastogi, V., Dasgupta, A., Das Sarma, A., & Sarlós, T. (2013). Optimal hashing schemes for entity matching. In *Proceedings of the 22nd international conference on world wide web* (pp. 295–306).

Demartini, G., Difallah, D.E., & Cudré-Mauroux, P. (2013). Large-scale linked data integration using probabilistic reasoning and crowdsourcing. *The VLDB Journal*, *22*(5), 665–687.

Dong, X., Halevy, A., & Madhavan, J. (2005). Reference reconciliation in complex information spaces. In *Proceedings of the 24th SIGMOD international conference on management of data* (pp. 85–96): ACM.

Euzenat, J., Ferrara, A., van Hague, W.R., Hollink, L., Meilicke, C., Nikolov, A., Scharffe, F., Shvaiko, P., Stuckenschmidt, H., Sváb-Zamazal, O., & dos Santos, C.T. (2011). Final results of the ontology alignment evaluation initiative 2011. In *Proceedings of the 6th workshop on ontology matching* (pp. 85–113).

Ferrara, A., Nikolov, A., & Scharffe, F. (2011). Data linking for the semantic web. *Semantic Web and Information System*, *7*(3), 46–76.

Gale, D., & Shapley, L.S. (1962). College admissions and the stability of marriage. *American Mathematical Monthly*, *96*(1), 9–15.

Hall, R., Sutton, C., & McCallum, A. (2008). Unsupervised deduplication using cross-field dependencies. In *Proceedings of the 14th SIGKDD conference on knowledge discovery and data mining* (pp. 310–317): ACM.

Hernández, M.A., & Stolfo, S.J. (1995). The merge/purge problem for large databases. *ACM SIGMOD Record*, *24*, 127–138.

Hogan, A., Zimmermann, A., Umbrich, J., Polleres, A., & Decker, S. (2012). Scalable and distributed methods for entity matching, consolidation and disambiguation over linked data corpora. *Web Semantics: Science, Services and Agents on the World Wide Web*, *10*, 76–110.

Hu, W., Chen, J., & Qu, Y. (2011). A self-training approach for resolving object coreference on the semantic web. In *Proceedings of the 20th international conference on world wide web* (pp. 87–96).

Hu, W., Yang, R., & Qu, Y. (2014). Automatically generating data linkages using class-based discriminative properties. *Data & Knowledge Engineering*, *91*, 34–51.

Isele, R., & Bizer, C. (2012). Learning expressive linkage rules using genetic programming. *The VLDB Journal*, *5*(11), 1638–1649.

Isele, R., & Bizer, C. (2013). Active learning of expressive linkage rules using genetic programming. *Web Semantics: Science, Services and Agents on the World Wide Web*, *23*, 2–15.

Isele, R., Jentzsch, A., & Bizer, C. (2011). Efficient multidimensional blocking for link discovery without losing recall. In *Proceedings of the 14th SIGMOD workshop on the web and databases*.

Kejriwal, M., & Miranker, D.P. (2013). An unsupervised algorithm for learning blocking schemes. In *Proceedings of the 13th international conference on data mining* (pp. 340–349): IEEE.

Kejriwal, M., & Miranker, D.P. (2015). Semi-supervised instance matching using boosted classifiers. In *Proceedings of the 12th extended semantic web conference. LNCS*, (Vol. 9088 pp. 388–402): Springer.

Kirsten, T., Kolb, L., Hartung, M., Groß, A., Köpcke, H., & Rahm, E. (2010). Data partitioning for parallel entity matching. *Proceedings of the VLDB Endowment*, 3.

Köpcke, H., & Rahm, E. (2010). Frameworks for entity matching: a comparison. *Data & Knowledge Engineering*, *69*(2), 197–210.

Köpcke, H., Thor, A., & Rahm, E. (2010). Evaluation of entity resolution approaches on real-world match problems. In *Proceedings of the VLDB endowment*, (Vol. 3 pp. 484–493): VLDB Endowment.

Koudas, N., Sarawagi, S., & Srivastava, D. (2006). Record linkage: similarity measures and algorithms. In *Proceedings of the 25th SIGMOD international conference on management of data* (pp. 802–803): ACM.

Levenshtein, V.I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, (Vol. 10 pp. 707–710).

Li, J., Tang, J., Li, Y., & Luo, Q. (2009). RiMOM: a dynamic multistrategy ontology alignment framework. *IEEE Transactions on Knowledge and Data Engineering*, *21*(8), 1218–1232.

Li, W.S., & Clifton, C. (2000). SEMINT: A tool for identifying attribute correspondences in heterogeneous databases using neural networks. *Data Knowledge and Engineering*, *33*, 49–84.

Locoro, A., David, J., & Euzenat, J. (2014). Context-based matching: design of a flexible framework and experiment. *Journal on Data Semantics*, *3*(1), 25–46.

McCallum, A., Nigam, K., & Ungar, L.H. (2000). Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the 6th SIGKDD conference on knowledge discovery and data mining* (pp. 169–178): ACM.

Mendes, P.N., & Jakob, M. (2011). García-silva, A., Bizer, C.: Dbpedia spotlight: shedding light on the web of documents. In *Proceedings of the 7th international conference on semantic systems* (pp. 1–8): ACM.

Mishra, S., Gandhi, T., Arora, A., & Bhattacharya, A. (2013). Efficient edit distance based string similarity search using deletion neighborhoods. In *Proceedings of the 16th joint EDBT/ICDT workshops on string similarity* (pp. 375–383): ACM.

Ngomo, A.C.N., & Auer, S. (2011). LIMES: A time-efficient approach for large-scale link discovery on the web of data. In *Proceedings of the 22nd international joint conference on artificial intelligence* (pp. 2312–2317).

Ngomo, A.C.N., & Lyko, K. (2012). EAGLE: Efficient Active learning of link specifications using genetic programming. In *Proceedings of the 9th extended semantic web conference. LNCS*, (Vol. 7295 pp. 149–163): Springer.

Ngomo, A.C.N., & Lyko, K. (2013). Unsupervised learning of link specifications: Deterministic vs. non-deterministic. In *Proceedings of the 8th workshop on ontology matching* (pp. 25–36).

Ngomo, A.C.N., Lehmann, J., Auer, S., & Höffner, K. (2011). RAVEN - active learning of link specifications. In *Proceedings of the 6th workshop on ontology matching* (pp. 25–36).

Nguyen, K., & Ichise, R. (2015a). Heuristic-based configuration learning for linked data instance matching. In *Proceedings of the 5th joint international semantic technology conference. LNCS*, (Vol. 9544 pp. 56–72): Springer.

Nguyen, K., & Ichise, R. (2015b). ScSLINT: Time and memory efficient interlinking framework for linked data. In *Proceedings of the 14th international semantic web conference posters and demonstrations track*.

Nguyen, K., Ichise, R., & Le, B. (2012a). Interlinking linked data sources using a domain-independent system. In *Proceedings of the 2nd joint international semantic technology. LNCS*, (Vol. 7774 pp. 113–128): Springer.

Nguyen, K., Ichise, R., & Le, H.B. (2012b). Learning approach for domain-independent linked data instance matching. In *Proceedings of the SIGKDD 2nd workshop on mining data semantics* (pp. 7–15): ACM.

Nikolov, A., d'Aquin, M., & Motta, E. (2012). Unsupervised learning of link discovery configuration. In *Proceedings of the 9th extended semantic web conference. LNCS*, (Vol. 7295 pp. 119–133): Springer.

Niu, X., Rong, S., Zhang, Y., & Wang, H. (2011). Zhishi.links results for OAEI 2011. In *Proceedings of the 6th workshop on ontology matching* (pp. 220–227).

Papadakis, G., Ioannou, E., Niederée, C., & Fankhauser, P. (2011). Efficient entity resolution for large heterogeneous information spaces. In *Proceedings of the 4th international conference on web search and data mining* (pp. 535–544): ACM.

Papadakis, G., Ioannou, E., Palpanas, T., Niederée, C., & Nejdl, W. (2013). A blocking framework for entity resolution in highly heterogeneous information spaces. *IEEE Transactions on Knowledge and Data Engineering*, *25*(12), 2665–2682.

Papadakis, G., Papastefanatos, G., & Koutrika, G. (2014). Supervised meta-blocking. In *Proceedings of the VLDB endowment*, (Vol. 7 pp. 1929–1940): VLDB Endowment.

Papadakis, G., Svirsky, J., Gal, A., & Palpanas, T. (2016). Comparative analysis of approximate blocking techniques for entity resolution. *Proceedings of the VLDB Endowment*, 9.

Pernelle, N., Saïs, F., & Symeonidou, D. (2013). An automatic key discovery approach for data linking. *Web Semantics: Science, Services and Agents on the World Wide Web*, *23*, 16–30.

Rahm, E., & Do, H.H. (2000). Data cleaning: problems and current approaches. *IEEE Data Engineering Bulletin*, *23*(4), 3–13.

Robertson, S.E., Walker, S., Jones, S., Hancock-Beaulieu, M.M., & Gatford, M. (1994). Okapi at TREC-3. In *Proceedings of the 3rd text retrieval conference* (pp. 109–123).

Rong, S., Niu, X., Xiang, W.E., Wang, H., Yang, Q., & Yu, Y. (2012). A machine learning approach for instance matching based on similarity metrics. In *Proceedings of the 11th international semantic web conference. LNCS*, (Vol. 7649 pp. 460–475): Springer.

Sarawagi, S., & Bhamidipaty, A. (2002). Interactive deduplication using active learning. In *Proceedings of the 8th SIGKDD conference on knowledge discovery and data mining* (pp. 269–278). New York, USA: ACM.

Sheila, T., Knoblock, C., & Minton, S. (2002). Learning domain-independent string transformation weights for high accuracy object identification. In *Proceedings of the 8th SIGKDD conference on knowledge discovery and data mining* (pp. 350–359): ACM.

Song, D., & Heflin, J. (2011). Automatically generating data linkages using a domain-independent candidate selection approach. In *Proceedings of the 10th international semantic web conference. LNCS*, (Vol. 7031 pp. 649–664): Springer.

Soru, T., & Ngomo, A.C.N. (2013). Rapid execution of weighted edit distances. In *Proceedings of the 8th workshop on ontology matching* (pp. 1–12).

Soru, T., & Ngomo, A.C.N. (2014). A comparison of supervised learning classifiers for link discovery. In *Proceedings of the 10th international conference on semantic systems* (pp. 41–44): ACM.

Suchanek, F.M., Abiteboul, S., & Senellart, P. (2011). PARIS: probabilistic alignment of relations, instances, and schema. *The VLDB Journal*, *5*(3), 157–168.

Thor, A., & Rahm, E. (2007). MOMA-a mapping-based object matching system. In *Proceedings of the 3rd biennial conference on innovative data systems research* (pp. 247–258).

Urbani, J., Kotoulas, S., Maassen, J., Van Harmelen, F., & Bal, H. (2010). OWL Reasoning with webpie: calculating the closure of 100 billion triples. In *Proceedings of the 7th european semantic web conference. LNCS*, (Vol. 5554 pp. 213–227): Springer.

Vesdapunt, N., Bellare, K., & Dalvi, N. (2014). Crowdsourcing algorithms for entity resolution. In *Proceedings of the VLDB endowment*, (Vol. 7 pp. 1071–1082): VLDB Endowment.

Volz, J., Bizer, C., Gaedke, M., & Kobilarov, G. (2009). Discovering and maintaining links on the web of data. In *Proceedings of the 8th international semantic web conference. LNCS*, (Vol. 5823 pp. 650–665): Springer.

Whang, S.E., & Garcia-Molina, H. (2014). Incremental entity resolution on rules and data. *The VLDB Journal*, *23*, 77–102.

Winkler, W.E. (2006). Overview of record linkage and current research directions. Tech. rep., Bureau of the Cencus.