Instance Matching for Large Scale Heterogeneous Repositories

Khai Hoang NGUYEN

Doctor of Philosophy

Department of Informatics
School of Multidisciplinary Sciences
SOKENDAI (The Graduate University for Advanced Studies)

Instance Matching for Large Scale Heterogeneous Repositories

Author:
Khai Hoang NGUYEN

Supervisor:
Prof. Ryutaro ICHISE

Doctor of Philosophy

Department of Informatics
School of Multidisciplinary Sciences
SOKENDAI (The Graduate University for Advanced Studies)

September 2016

A dissertation submitted to Department of Informatics,
School of Multidisciplinary Sciences,
SOKENDAI (The Graduate University for Advanced Studies),
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Advisory Committee

1. Assoc.Prof. Ryutaro Ichise	SOKENDAI
2. Prof. Hideaki Takeda	SOKENDAI
3. Prof. Ken Satoh	SOKENDAI
4. Prof. Akiko Aizawa	National Institute of Informatics
5. Assoc.Prof. Naoki Fukuta	Shizuoka University

Acknowledgements

I would like to express the deep gratitude to my advisor Prof. Ryutaro Ichise for the perfect supervision. His immense contribution of inspiration, encouragement, and patience are the priceless values and the reasons for the excitement and productivity of my research.

I am thankful to my advisory committee, Prof. Hideaki Takeda, Prof. Ken Satoh, Prof. Akiko Aizawa, Prof. Naoki Fukuta, Prof. Yamada Seiji, and Prof. Ikki Ohmukai for their insightful comments and brilliant suggestions.

I appreciate my lab members for being willing to discuss and provide thoughtful comments on my research. Also, the sharing from all dear friends in NII also delivered a warm encouragement.

I am grateful to my family for always being by my side with the constant source of love and motivation.

Lastly, I would like to thank SOKENDAI, National Institute of Informatics, and Vietnamese Government for the educational and financial support.

SOKENDAI (THE GRADUATE UNIVERSITY FOR ADVANCED STUDIES)

School of Multidisciplinary Sciences
Department of Informatics

Doctor of Philosophy

Instance Matching for Large Scale Heterogeneous Repositories

by Khai Hoang NGUYEN

Abstract

The recent years have witnessed the fastest period of the development of digital information. Individuals and organizations are publishing data at the highest acceleration. Accompanying with the immense amount of data are many challenges. Among them, instance matching, which identifies different instances of the same entity (aka. coreferences) in various data sources, has been considered as a critical problem. The reason is that the independently created instances are usually incomplete, because of the inconsistency nature of data publication (e.g., purpose, tool, user, and scale). Instance matching helps not only to collect the multiple aspects of entities but also to improve the consistency and non-redundancy of the data.

The dissertation summarizes our contributions to several issues of instance matching. First, we focus on scalability, which is very important for deploying large matching tasks. We develop a time and memory efficient framework named $ScSLINT$. $ScSLINT$ is a specification-based framework. It generates coreferences on the basis of given instructions, such as matching properties, similarity metrics, and filtering strategy. $ScSLINT$ promotes the matching task to at least 10 times faster compared to state-of-the-art frameworks. $ScSLINT$ is also the unique framework successfully tested on quadrillion scale dataset using a memory-limited machine. Then, based on the architecture of $ScSLINT$, further systems and algorithms have been introduced.

We propose systems and algorithms for two scenarios of instance matching: supervised and non-supervised. These scenarios are different at the presence of training data. For supervised matching, we propose a specification-based system and a feature to enhance classification-based systems.

For non-supervised instance matching, we propose ASL, a schema-independent instance matching system for linked data. Because of the inconsistency in the schemas of different

repositories, it is important to develop a general system that can work with any repository with any schema. *ASL* finds the equivalent properties and constructs the matching specification. Experiments on 246 datasets with different schemas and domains show that *ASL* obtains high accuracy and significantly improves the quality of discovered coreferences against the previous systems.

For supervised instance matching, we propose *ScLink* system and *R2M* ranking feature. *ScLink* is a system for specification-based matching. The most important part of a specification-based system is the construction of specification. Existing specification learning algorithms are either ineffective or inefficient. Furthermore, there is space for improvement of scalability as previous systems have not optimized the candidate generation step. *ScLink* is the combination of two novel algorithms *cLearn* and *minBlock*. *cLearn* finds the optimal matching specifications by detecting high-quality equivalent properties and optimizing similarity metrics. *minBlock* enhances the important blocking step of the matching process. This algorithm restricts the matching task into a compact subset instead of the huge pairwise alignments between data sources. In addition, *ScLink* employs a novel string similarity metric, *Modified-BM25*, which aims at the better disambiguation against the existing metrics. We evaluate *ScLink* using 15 standard matching tasks on relational databases and linked data. The experiment results show that *cLearn* significantly increases F1 score compared to existing specification learning algorithms. Meanwhile, *minBlock* discards up to 95% of unnecessary candidates and therefore considerably contributes to the reduction of processing time. *Modified-BM25*also shows its usefulness on real datasets.

While the specification-based instance matching is good at scalability, the classification-based approach has the advantage of generalization based on the solid theory of machine learning. We also approach the problem of instance matching in classifier-based fashion. We find that the limitation of usual classifiers is the ignorance of ranking the coreferences, an important factor of instance matching. We propose ranking feature *R2M* for classification-based matching systems. *R2M* significantly improves the quality of trained models and advances them to remarkably outperform *ScLink* as well as existing classifier-based matching systems. We also compare the performance of the proposed systems and the classifier with *R2M* ranking feature. We show that the usage of our systems and algorithms depends on the matching task and should be considered under the trade-off between accuracy and scalability.

Contents

List of Figures

List of Tables

Symbols

R	repository
R_S	source repository
R_T	target repository
$v{=}p(x)$	value of instance x at property p
$\langle x, y \rangle$	instance pair formed by instance x and y
$\langle p_S, p_T \rangle$	property mapping formed by property p_S and p_T
A	actual coreferences
I	detected coreferences
C	set of candidate
C_L	labeled candidates
C_U	unlabeled candidates
R_{SL}	part of R_S for generating C_L
R_{SU}	part of R_S for generating C_U
M	set of property mappings
B	blocking model
B_{def}	default blocking model
B_{opt}	optimal blocking model
S	matching specification
S_{opt}	optimal specification
F	set of similarity functions
F_{def}	set of initial similarity functions
G	aggregation function
G_{inp}	list of input aggregation function
D	determiner
E	preprocessing function
w	string token

σ	similarity metric
δ	similarity function
$\delta^{\sigma}_{\langle p_S, p_T \rangle}$	specific similarity function using σ metric on property mapping $\langle p_S, p_T \rangle$
λ	threshold of boolean aggregation function
ω	threshold of stable filter

Chapter 1

Introduction

In this chapter, we begin with the motivation of instance matching problem and our study (Section 1.1). After that, we summarize our contributions to different aspects of this problem (Section 1.2). Finally, we briefly introduce the remaining chapters of the dissertation (Section 1.3).

1.1 Background

The Internet nowadays plays an important role in almost every aspect of our activities. It has profoundly changed the way we create and collect the information. The current state of technology allows us to easily access the Internet and efficiently look up the information. For that reason, the Internet is widely believed that it is able to provide everything. However, even living in an era of digital data overwhelmed like this, not all people's needs are really satisfied by the current technologies.

As an example, let's consider a powerful tool like search engines. Search engines provide the mechanism of indexing web pages and answering user's query. One can use search engines to get the information of any entity, on any topic. However, the information that search engines can offer is only facts, not knowledge. Therefore, search engines usually give perfect results to concrete queries but fail to response to complex queries. For example, they return the correct pages of the query 'Tokyo' but may fail to answer the query 'capital of the most developed Asian country'. It is because of two issues. First, the pages describing these information are different and none of them contains everything of the entities mentioned in the query. Second, the pair 'Tokyo' and 'the capital of Japan', and the pair 'Japan' and 'the most developed Asian country', are not realized to be coreferent to the same entities. The query can be satisfied by resolving any of those issues. However, a solution for the first issue is impossible because it originates from the nature of the Web, in which pages are distributed, independently created, and usually incomplete. In order to leverage the extensively resourceful data without compromising the data publication, instance matching is used as a solution for the second issue. Another example is the integration of the Web pages resulted from querying an interested entity (e.g. a product, movie, or city) on the Internet. It is very time-consuming for a user to browse all the returned results of the search engine to construct the complete information of an entity. In this situation, instance matching is a powerful tool for removing the duplications and collecting multitude aspects of the entity.

Abovementioned is a motivating example of instance matching for the mentions in Web pages. In broad terms, instance matching is to detect different instances of the same real-world entities. Instance matching is an important task in knowledge discovery and data mining, especially in *data integration* [46, 126, 146], *data cleansing* [36, 129] and *linked data interlinking* [40]. In *data integration* and *data cleansing*, instance matching is used to detect the duplicated, erroneous, and outdated data. By applying instance matching, the integrated data is guaranteed to be consistent in term of data representation (e.g., data format and schema). Moreover, in many databases, the duplicates are not allowed. Therefore, instance matching also guarantees the integrity because it keeps the

data clean. In *linked data*, instance matching is an essential component for linking the coreferent instances of different repositories. There are many relationships between the instances in the Web of linked data (e.g., class, sub-class, property, and sub-property). However, the owl:sameAs links, which connect the coreferences, are the most important because they can be used to discover other relationships. In short, instance matching enriches the knowledge of *linked data* by connecting the independently created *"data silos"*. For its importance, instance matching has been extensively studied. However, the achievement of an ultimate solution is still an open research problem.

As introduced above, our study is motivated by the problem of *data integration* in general and *linked data interlinking*. We consider the problem of instance matching in the scope of (semi-)structured repositories, where instances are represented as a set of property-value pairs. Examples for this type of repository are *relational data* and *linked data*. In *relational data*, each row represents an instance, whose property-value pairs are the cells in that row. In *linked data*, an instance is represented by a set of RDF triples. Each RDF triple contains three elements: *subject*, *predicate*, and *object*. A *subject* is usually the URL of the instance. A *predicate* is equivalent to a property in relational data. An *object* is the value described by a particular predicate.

The challenges of instance matching are different between types of the repository (e.g., relational data, linked data, text-based data). However, an instance matching algorithm generally has to cope with three difficulties: *heterogeneity*, *ambiguity*, and *scalability*. Solving the problem of instance matching is equivalent to dealing with these challenges. *We are motivated from modest contributions to the solution for each of them.*

- **Heterogeneity:** The issue of heterogeneity is very common in instance matching. It occurs in both of the *schemas* and *instances*. A schema contains the description of all properties used in a repository. The heterogeneity of schemas arises from the fact that different repositories do not use the same schema. Meanwhile, the heterogeneity of instances originates from the inconsistent representations and the sources of fact where the data is collected. An example of heterogeneity is illustrated in Figure 1.1. In this figure, two *linked data* instances of the place 'Harlem' are depicted, one is from *Freebase*[1] repository and the other is from *NY-Times*[2] repository. It can be realized that the properties and values stored in these instances are different. In practice, the same attributes of an entity can be declared by various properties and represented in different formats (e.g., measurement unit, name ordering, and language) or even different values (e.g., 'New York' and 'NYC', 'fridge' and 'refrigerator'). Basically, any instance matching algorithm

[1] http://rdf.freebase.com/ns/en.harlem
[2] http://data.nytimes.com/harlem_nyc_geo

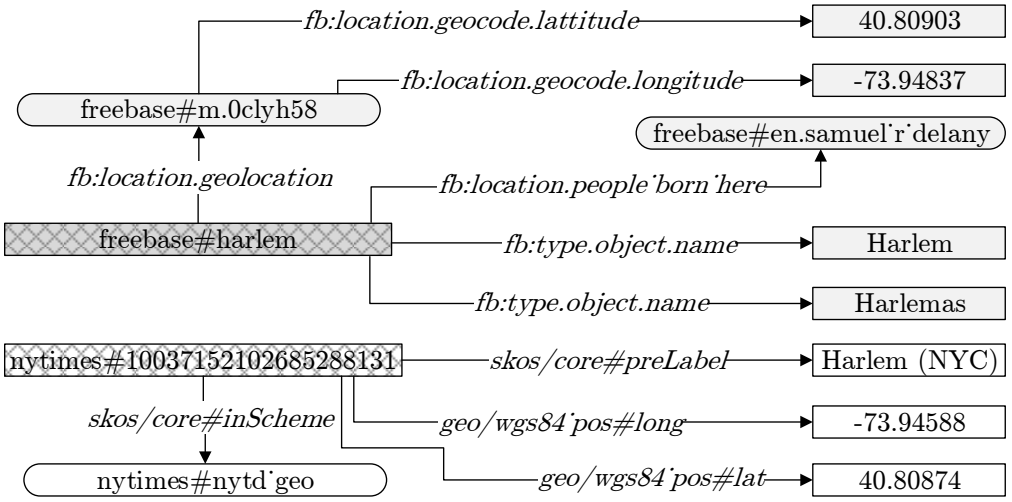


FIGURE 1.1: An example of heterogeneity.

has to compare the instances and since then, the heterogeneity becomes a hard obstacle.

The early state of instance matching is the manually operated systems, in which the instances comparisons are specified given domain and particular repositories (e.g., focusing on the geographic coordinate of locations and telephone number of companies). Although this approach achieved acceptable matching results, an expert's knowledge may not cover various domains or it is time-consuming to specify the useful information. Consequently, the manual system becomes difficult to use for all users and in all domains. *We focus on the heuristic-based construction of specification that does not depend on the domain and thus is applicable to any repository, related to any domain.*

- **Ambiguity:** Different values can describe the same fact, and contradictorily, the same value can simultaneously refer to different facts. That is the issue of ambiguity. For example, 'Isoaai' is named for 491 places in Finland and there are about 1,724 locations sharing the name 'San José'. Another example is two instances in DBpedia: 'Channel Islands National Park'[3] and 'Channel Islands of California'[4]. Although these instances describe different places, it is difficult to recognize that even when presenting them to a human who has not known the fact in advance. Together with heterogeneity, ambiguity is the reason of the limitation in the accuracy of most instance matching algorithms.

[3] http://dbpedia.org/page/Channel_Islands_of_California
[4] http://dbpedia.org/page/Channel_Islands_National_Park

The most important part of disambiguation is to successfully estimate the similarity of different values, especially for string literals. To this aspect, most instance matching systems relied on traditional string similarities, linear combinations of various metrics, and unrelated to the characteristics of given repositories. This manner eventually ignores the advantage of each similarity metric in different specific cases. *We study different manners of similarity metrics combination as well as optimization methods for selecting appropriate metrics given the interested repositories.*

- **Scalability:** Finally, scalability is also an important issue that needs to be taken into account, considering 2.5 quintillion bytes of data created daily (a statistic of 2015). A single repository may contain millions of instances and therefore, a trillion scale instance matching problem is possible to be frequently demanded. However, most of the previous work simply checked the pairwise instances of given repositories and complied an infeasible matching task for large repositories [6, 45, 147]. *We study the algorithms that restrict the comparisons into a compact space for reducing the computational complexity and thus scale up the system's capability.*

  In addition, even with the support of current technology, existing frameworks are not fully optimized for really high-scale matching tasks. The main reason is that they reused existing but complicated index structures, which is not designed for the particular instance matching task. *We are interested in developing a time and memory efficient framework that can handle very large repositories.*

  Furthermore, when the repository's scale increases, the heterogeneity and ambiguity are likely to be more complicated. This situation is obviously expressed on linked data, which is considered to be large and much heterogeneous. For example, consider the DBpedia 3.9, there are 45,858 unique properties, 5,314,053 of only non-redirected instances[5], and more than 20,000 instances sharing the token 'John' or 'William'. Freebase is even more than that. About 2.5 million predicates are used in Freebase and this number is at least 50 times larger than those of DBpedia, the main hubs of the *linking open data*. Large scale repositories nowadays mostly accompany with open data, which offers the right of creating and editing to multiple users. On the one hand, this mechanism provides the irrefutable advantages for the collection of immense data. On the other hand, it makes the repositories more vulnerable to the inclusion of incorrect and inconsistent information. To this end, scalability involves not only computational issue but also accuracy issue.

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

There are two main approaches for instance matching: specification-based and classification-based.

- For specification-based approach, a system estimates the matching score of instances and find the coreferences based on the scores. The score estimation and utilization are described by a specification. For example, it includes the properties mappings, similarity metrics, and matching thresholds. Most researches on specification-based matching are about the construction of the specification. As earlier discussion, the manual method constructs the specification with the help of human experts and is restricted to well-known domains. One of our focuses is the heuristic-based construction of specification. Furthermore, considering instance matching in supervised scenario, when some curated coreferences are given, *we study the problem of elaborate and automatic specification construction.*
- For classification-based approach, instance matching is treated as a classification problem. The classifier is trained using labeled coreferences and is applied to predict the coreferent state (positive/negative) of given instance pairs. An unresolved problem of previous classification-based matching systems is the ranking of instances pairs. Usual classifiers do not consider the variety of ambiguities for different values, which is very important in instance matching. For example, considering the fact that there are many persons sharing the name 'William' rather than 'Specter'. A very high matching score is expected for matching instances of a 'William' in order to confidently determine a positive coreference. However, for a 'Specter', a medium matching score may be convincing. That is, the ranking on matching scores of the instances sharing differently ambiguous values may enhance the accuracy. *We finally investigate the problem of ranking in the context of classification-based instance matching.*

1.2 Contributions

The dissertation is motivated from the above challenges of heterogeneity, ambiguity, and scalability. We propose a framework, algorithms, systems, and some components to solve the challenges. Our work focuses on two different scenarios of instance matching: *non-supervised*[6] and *supervised*. Although *non-supervised* matching has wider applicability as it does not require training data, when the matching quality is prioritized and the training data is available, a *supervised* method is the best option to improve the accuracy. Therefore, we also focus on the supervised scenario. For *non-supervised*,

[6] We use the term *non-supervised* to discriminate from *unsupervised*. *Non-supervised* only means that the curated data is unavailable. *Unsupervised* conventionally implies machine learning is applied.

TABLE 1.1: Summary of the contributions.
Legends: sp: *Supervised*, n-sp: *Non-supervised*
H: *Heterogeneity*, A: *Ambiguity*, S:*Scalability*

Contribution	sp	n-sp	H	A	S
1. *ScSLINT* framework	✓	✓			✓
2. *ASL* system		✓	✓		✓
Heuristic-based property mappings			✓		
Token-based property-driven blocking strategy					✓
3. *ScLink* system	✓		✓	✓	✓
minBlock blocking model optimization	✓		✓		✓
cLearn specification optimization	✓		✓	✓	
Modified-BM25 string similarity metric	✓	✓		✓	
4. *R2M* ranking feature	✓		✓	✓	

we propose *ASL*, a schema-independent instance matching system. The main advantage of *ASL* is that it can work on any repository with any schema. For *supervised*, we propose *ScLink* and *R2M*, which focus on specification-based and classification-based instance matching, respectively. Specification-based matching has the advantage in scalability while classification-based matching is expected to has better generalization ability. *ScLink* is a system that can find the optimal matching specification by using the training data. *R2M* is a ranking feature for enhancing the accuracy of classifiers. In addition, we develop *ScSLINT* framework for processing large-scale instance matching tasks. For each system, we install multiple novel components. Those contributions, as well as their respective targets, are listed in Table 1.1. Here we briefly describe those contributions.

1. *ScSLINT* [107]. *ScSLINT* is an efficient instance matching framework. *ScSLINT* is designed for general instance matching tasks and is applicable to various types of data. *ScSLINT* aims at scalable instance matching. It is optimized with *parallel processing* technology and *light-weight indexing* structures. *ScSLINT* contains multitude of built-in functions, including the indexers for different sub-tasks of instance matching. *ScSLINT* can be used for both supervised and non-supervised matching tasks. Based on *ScSLINT*, we further develop two systems, *ASL* and *ScLink*.

2. *ASL* [108]. *ASL* is a schema-independent instance matching system. This system focuses on the challenges of *heterogeneity of schemas* and scalability. The most prominent feature of *ASL* is the introduction of a *heuristic-based* strategy to find the equivalent properties. Such strategy allows the system to work with any repository and with any schema. For scalability, we focus on the reduction of comparisons. We try to select only the potential instances pairs from the pairwise

alignments of the input repositories. In order to achieve this goal, we apply a *token-based property-driven* blocking strategy.

3. *ScLink* [111]. *ScLink* is a scalable, supervised, and specification-based system. The targets of *ScLink* consist of all the challenges: heterogeneity, ambiguity, and scalability. In *ScLink*, we propose two supervised algorithms, *cLearn* [103, 105, 106, 109] and *minBlock*. *cLearn* applies an apriori-like heuristic for finding the most suitable property mappings and similarity metrics. To this end, it solves the problem of heterogeneity and ambiguity in supervised scenario. Meanwhile, *minBlock* focuses on the scalability. It searches for a blocking model, which aims at optimally reducing the pair-wise alignments of instances between the input repositories. Comparing to the blocking strategy used in *ASL*, *minBlock* is expected to deliver much better performance. We also present a *Modified-BM25* metric to estimate the string similarity with a disambiguation ability. *Modified-BM25* is a combination of multiple state-of-the-art metrics, in order to consider different aspects of the interested strings.

4. *R2M* [110]. *R2M* is a ranking feature for classification-based instance matching systems. In order to include the ranking factor as the prior discussion in Section 1.1, we propose *R2M* feature for enhancing the classification-based matching systems. The basic idea is to train the model of *R2M* estimation by a learning to rank algorithm. After that, the *R2M* feature is computed for all examples. The final classifier is built upon the original as well as the newly created feature. In general, *R2M* is designed for classifiers and thus it contributes to the solution for heterogeneity and ambiguity.

1.3 Outline

The next chapters of the dissertation are summarized as follows.

- **Chapter 2:** We discuss the background knowledge of instance matching and its basic issues. After that, we review the work related to several aspects of instance matching, including frameworks, systems, and algorithms. We also categorize existing instance matching systems using various criteria.
- **Chapter 3:** This chapter describes *ScSLINT* framework. In this chapter, we detail the general architecture of *ScSLINT*, as well as its components such as the indexing, supported similarity metrics, and similarity aggregations. We report the experiments that evaluate *ScSLINT* on very large datasets and compares *ScSLINT* with prior frameworks.

- **Chapter 4:** This chapter describes *ASL* system. We detail the schema-independent mechanism of *ASL* and the token-based property-driven blocking strategy. Experiments on 253 datasets to evaluate the schema-independent ability and the performance of the blocking technique are reported.

- **Chapter 5:** This chapter describes the scalable supervised instance matching system *ScLink*. We detail the two supervised algorithms, *cLearn* and *minBlock*. We also introduce the *Modified BM25* string similarity metric. We report several experiments that evaluate *cLearn*, *minBlock*, and the comparisons to other supervised algorithms and systems. The performance of *Modified-BM25* and the necessary amount of training data are also examined.

- **Chapter 6:** This chapter describes the ranking feature *R2M* for classification-based instance matching systems. We detail *R2M* and present the fast optimization method for *R2M* model. We report the comparisons between the baseline classifiers and the ones which are trained with *R2M* feature. The comparisons to previous systems are also reported.

- **Chapter 7:** This chapter summarizes the dissertation and discusses our perspectives on the potential work to further advance the capability of current solutions to instance matching.

Chapter 2

Background and related work

We begin this chapter with the basic workflow of instance matching systems (Section 2.1). Following that is the description of the input (Section 2.2) and other components of the instance matching process (Section 2.3 and 2.4). After that, we describe the evaluation metrics for instance matching systems (Section 2.5). The chapter also covers the literature review and addresses the unresolved problems (Section 2.6 and 2.7).

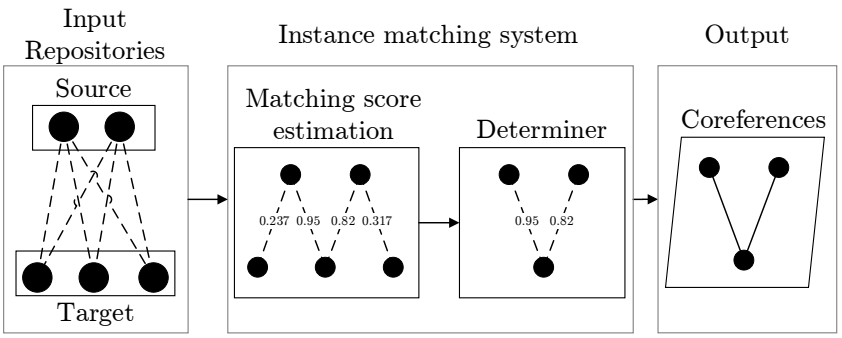


FIGURE 2.1: The basic workflow of instance matching.

2.1 The basis of instance matching

Instance matching is to detect the coreferent instances from input repositories: the source and the target. Concretely, the problem is stated as follows. Given two repositories R_S and R_T, which are the source and the target, respectively. The mission is to find the subset A of the Descartes product $R_S \times R_T$, so that each element of A is a pair of coreferent instances, and its complement $R_S \times R_T \setminus A$ does not contain any coreference.

The process of instance matching contains multiple components. In Figure 2.1, we illustrate a simple instance matching system with two main components: *Matching score estimation* and *Determiner*. The *matching score estimation* computes the score of pairwise alignments of all instances (the circles in Figure 2.1) in the source and the target repositories. The *determiner* takes those scores (and sometime all intermediate values) as the input and produces the final coreferences. There are various techniques for the *matching score estimation* as well as *determiner*. These two components are very important and always exist in any instance matching system, although they can be divided into smaller components. For example, many systems additionally install the *property alignment*, *blocking* to the *matching score estimation*. The *property alignment* finds the mappings between properties when the input repositories use different schemas. The *blocking* filters the pairwise instances and feeds only the potential coreferences into the *matching score estimation*. Blocking is very important for large datasets because it is computationally expensive to consider all pairwise instances.

Before discussing the basic techniques of *matching score estimation* and *determiner* (Section 2.3 and 2.4), we describe the input of instance matching.

2.2 Repository, schema, property, and instance

We study the problem of instance matching on structured and semi-structured repositories. A repository of these kind consists of two components: *schema* and *instances*. A *schema* specifies the properties used to describe the instances. Is also can be considered as the list of all properties. Each *property* is specified by *name*, *type*, and *range*. *name* is string and usually describe the exact function of the property. For example, *name* can be 'title', 'address', 'affiliation', etc. However, in some repositories, many properties are named with just an *id* number. This is one of the barriers of matching properties using *name*. *Type* specifies the data type of the values described by the property. For example, numeric, datetime, string, and URI are the most common data types. Type information is important for selecting the suitable similarity metrics in matching score estimation. The last information is *range*. *Range* restricts the values described by the property to a limited set. For example, the range of 'born in' can be defined as the set of 'cities'.

The main difference between structured and semi-structured data is the number of properties is fixed for all instances of structured data but not for semi-structured data. In other words, the number properties in each instance of semi-structured data may be different. In structured data, each instance is described by all properties, even when missing values are available. That is, for a semi-structured data, seeing all instances is necessary to draw the complete schema. In addition, the three information *name*, *type*, and *range* are always described for structured repositories, but not the semi-structured ones. In such cases, usually only *name* is found. Usually, for community-based repositories (e.g., Internet-based and crowdsourcing data), it is difficult to (consistently) specify the schema. Therefore, these kinds of repositories are usually semi-structured. One of our studied objects is the semi-structured *linked data*, whose most repositories are crowdsourcing data. One reason for us to select the *linked data* because it is considered as the data of the future technology and is growing at an impressive speed [140].

Considering the domain, there are also two types repository: *single domain* and *cross domains*. A single domain repository contains the instances of only one domain and a cross-domain repository may contains the instances of various domains. In the second case, a repository can be divided into different classes with respective to the domains. In our study, we consider the problem of instance matching on single domain repositories. For a cross-domain repository, we separate it into different repositories using the class information and treat them as single domain repositories.

The second component of a repository is the list of instances. An instance is a representation of an entity and is stored by using the schema. Concretely, for each instance, the

Properties

	Name	Hometown	Birth year	...
Instances	Elvis Presley	Mississippi	1935	...
	Michael Jackson	Indiana	1958	...
	Eminem	Missouri	1972	...
	...	...	...	...

FIGURE 2.2: An example of relational data.

	subject	predicate	object
Instance of Tokyo	db:Tokyo	rdfs:label	Tokyo Metropolis
	db:Tokyo	dbo:areaTotal	844.66 sq. mi
	db:Tokyo	dbo:population	13,506,607
	...	...	...
Instance of JMA	db:JMA	rdfs:label	Japan Meteorological Agency
	db:JMA	dbo:employees	5,539
	db:JMA	dbo:location	db:Tokyo
	...	...	...
Instance of Chiyoda	db:Chiyoda	rdfs:label	Chiyoda
	db:Chiyoda	dbo:areaTotal	4.50 sq. mi
	db:Chiyoda	dbo:population	54,462
	db:Chiyoda	dbo:prefecture	db:Tokyo
	...	...	...

FIGURE 2.3: An example of linked data.

value of each property is declared. Some researchers may call such *value* as *attribute*. In this dissertation, we use the term *value* for avoiding the confusion between *attribute* and *property*. Figure 2.2 and 2.3 illustrate some example instances of *relational data* and *linked data*, respectively. In *relational data*, each row represents an instance, whose property-value pairs are the cells in that row. In *linked data*, an instance is represented by a set of RDF triples. Each RDF triple contains three elements: *subject*, *predicate*, and *object*. A *subject* is usually the URL of the instance. A *predicate* is equivalent to a property in relational data. An *object* is the value described by a particular predicate. By representing instances by RDF triples, the number of triples for different instances are varied. This is one of the characteristics of the semi-structured data as discussed above.

The coreferences are the instances that co-refer to the same real-world entity. Such instances may exist among different repositories as well as the same repository. Some studies consider the instance matching within one repository as the problem of *data*

Table 2.1: Notation of inputs of instance matching.

Symbol	Meaning
R_S	source repository
R_T	target repository
p	property
$p(x)$	values of x at property p
o	value

deduplication [36, 129]. For the matching of different repositories, they assume the repositories are "clean". That means the coreferences do not exist within one repository. In our work, we study the problem of instance matching between different repository without that constraint of "cleanliness". Therefore, we treat the repository as a *multiset* of instances. Although a repository also consists of schema, it is usually denoted as only the instances and we also follow this manner.

Table 2.1 lists the notation of the inputs of instance matching. These notations are used consistently thorough the dissertation.

We described the input of instance matching. Next, we discuss the basic techniques of the *matching score estimation* and *determiner*.

2.3 Matching score estimation

Matching score estimation is an important step in instance matching. A matching score represents the similarity of two instances. The most basic form of matching score estimation is the aggregation of literal similarities. Literal similarities are the similarities of the values of the given instances. Therefore, the matching score estimation consists of two steps. The first step is the computation of literal similarities. The second step is the aggregation of the first step's results, in order to produce a final score.

Suppose that we need to compute the matching score of two *linked data* instances x and y of Figure 2.4. A good matching specification should contain the correct property mappings as follows: 'label' and 'name' (*string*), 'population' and 'population' (*numeric*), 'website' and 'homepage' (*URI*), and 'established' and 'established' (*datetime*). Then, for each property mapping, one or many similarity metrics are applied to compute the literal similarities. The final matching score of x and y is computed from those literal similarities.

In practice, the specification is manually or automatically determined. The variety of systems is also measured by how to construct or use the specification. In this section, we review the techniques of similarity metrics and similarity aggregation instead of the

	subject	predicate	Object (type)
Instance x	dba:Tokyo	label	Tokyo Metropolis (string)
	dba:Tokyo	areaTotal	844.66 sq. mi (numeric)
	dba:Tokyo	population	13,506,607 (numeric)
	dba:Tokyo	website	www.metro.tokyo.jp (URI)
	dba:Tokyo	established	1889/05/01 (datetime)
	...	...	...
Instance y	dbb:2378295317	name	Tokyo (string)
	dbb:2378295317	population	13,507,000 (numeric)
	dbb:2378295317	homepage	http://www.metro.tokyo.jp (URI)
	dbb:2378295317	established	May 1 1889 (datetime)
	...	...	...

FIGURE 2.4: Linked data instances with data types.

construction of the specification. We review the commonly used similarity metrics and aggregation strategies. The metrics are categorized into four groups with respective to four main data types. In case of fully structured data, data types are specified by the schema. However, when such specifications are missing, data type of each property is determined by the majority type of its object values.

2.3.1 Literal similarity

2.3.1.1 String

String is the most important data type in many problems, as major of data is text. In instance matching, *string* is also an essential types. Many primary attributes are represented by *string*, such as *name*, *label*, *description*, etc., and such attributes attaches to almost all entities.

Many string similarities have been proposed and can be divided into different approaches. The representative approaches consist of intersection-based, transformation-based, corpus-based, and hybrid similarities. A string can be decomposed into a *sequence* of tokens, characters, or n-gram. Existing string similarity metrics work with all of token, character, and n-gram inputs. The n-gram of a string s is defined as the sequence of n-length substring of s. Some metrics treat the strings as the original sequences, while some others treat the sequences as the *sets*. In order words, the duplicated elements are removed and their order is ignored. Next, we describe the widely used similarity

metrics. We use A and B to denote the decomposed form of the input strings. A and B can be either sets or sequences depending on the context.

1. **Intersection-based** metrics estimate the similarity based on the matching elements.

   - **Common substring** measures the ratio between the common parts of the given *sequences*.

     $$\sigma_{cms}(A, B) = \frac{2 \times lcs(A, B)}{|A| + |B|} \tag{2.1}$$

     where $lcs(A, B)$ is the *longest common subsequence* of A and B. There are three strategies for measuring $lcs(A, B)$: from the beginning of the input (prefix), from the end of the input (suffix), and just finding the longest common subsequence.

   - **Szymkiewicz-Simpson** (aka. overlap coefficient) is defined as the ratio of the intersection divided by the size of the smaller of the given *sets* [145].

     $$\sigma_{overlap}(A, B) = \frac{|A \cap B|}{min(|A|, |B|)} \tag{2.2}$$

   - **Jaccard similarity coefficient** (aka. Jaccard index) is defined as the ratio of the intersection divided by the size of the union of the input *sets* [64].

     $$\sigma_{jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{2.3}$$

   - **Sørensen index** (aka. Dice's coefficient) is defined as the ratio of the intersection divided by the sum of the size the input *sets* [31, 149].

     $$\sigma_{dice}(A, B) = \frac{2 \times |A \cap B|}{|A| + |B|} \tag{2.4}$$

2. **Transformation-based** metrics estimate the similarity based on the operations of transforming one of the inputs into the other.

   - **Levenshtein** The Levenshtein distance quantifies the minimum number of operations (insertion, deletion, and substitution) required to transform one sequence into the other [79]. The distance is defined as follows.

     $$\delta_{levenshtein}(A, B) = \min_{op_n(...op_1(A))} \sum_{i \in [1,n]} cost(op_i) \tag{2.5}$$

where $op_n(...op_1(A)) = B$ is a series of nested operations needed to transform A into B; and $cost(op_i)$ is the cost of performing the i^{th} operation. Levenshtein distance applies the same cost (equal to 1) for all type of operations. That is, for Levenshtein distance, the Equation 2.5 counts the number of operation.

Levenshtein distance can be used reversely to estimate how similar two *sequences* are. The frequently used Levenshtein-based similarity is the Levenshtein distance divided by the length of the longer sequence.

$$\sigma_{levenshtein}(A, B) = \frac{\delta(A, B)}{max(|A|, |B|)} \tag{2.6}$$

- **Jaro and Jaro-Winkler** are based on the vicinity of the common elements of the given sequences [65]. The Jaro distance is computed as follows.

$$\delta_{jaro}(A, B) = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3} \times (\frac{m}{|A|} + \frac{m}{|B|} + \frac{m - \frac{t}{2}}{m}) & \text{otherwise} \end{cases} \tag{2.7}$$

where m is the number of shared elements and t is number of transpositions. Two elements are considered as matching if they are the same and their index in the parent's sequences are not farther than $\frac{max(|A|,|B|)}{2} - 1$. The transposition is defined as the number of matching elements divided by 2.

The Jaro-Winkler [160] distance adds the factor of prefix common substring to Jaro distance. It is computed as follows.

$$\delta_{jaro-winkler} = \delta_{jaro}(A, B) + \ell p(1 - \delta_{jaro}(A, B)) \tag{2.8}$$

where ℓ is the length of the shared prefix. ℓ is bounded into 4 elements, p is a scaling factor for controlling the impact of the common prefix. p is normalized to be ≤ 0.25 in order to guarantee that the result is in the range of $[0, 1]$. In original work and many others, the default value of p is 0.1.

The complement of Jaro and Jaro-Winkler define the similarities of the respective distances.

$$\sigma_{jaro}(A, B) = 1 - \delta_{jaro}(A, B) \tag{2.9}$$

$$\sigma_{jaro-winkler}(A, B) = 1 - \delta_{jaro-winkler}(A, B) \tag{2.10}$$

3. **Corpus-based** metrics uses a corpus to support the estimation of similarity. The corpus can be the structured dictionary, taxonomy, or just raw text, on which

some statistical factors can be observed.

- **WordNet-based similarities** are the metrics based on WordNet database, a lexical database of English language [94]. WordNet contains the hierarchical relationships and the synonyms of words. Therefore, the similarities using WordNet are expected to capture the semantic relatedness of the input strings. Using WordNet, the inputs are the tokens and a token is called concept.

  Many metrics have been proposed for using WordNet [91]. The widely used metrics are Resnik [131], Lin [82], JCN [67], LCH [78], WUP [162], HSO [52], LESK [7], and VECTOR [123]. The first three metrics are based on the common subsumer (i.e., parent nodes) of the concepts. The next three metrics uses the relationship structure (e.g., length of path and depth of nodes). The LESK metric makes use of the WordNet's gloss of concepts. It computes the similarity based on the overlaps between the pairwise glosses of the given concepts and their hyponyms. VECTOR creates the vector representation for each concept and use vector space's metrics for computing the similarity. This vector is the average of the co-occurrence vectors representing the glosses. The co-occurrence vector of the word w is constructed by concatenating the co-occurrence of w and other words within a given corpus.

- ***TF-IDF cosine*** is one of the most commonly used metrics. This metric consists of two parts: the *TF-IDF* [136] and the *cosine* similarity. *TF-IDF* is a weighting scheme measure the importance of words (i.e., term) based on their frequency. The original context of *TF-IDF* is the *information retrieval* problem. In that problem, the corpus is a set D of text documents and each document contains many terms. *TF* (term frequency) $tf(t, d)$ reflects the number of times that the term t appears in the document d. *IDF* (inverse document frequency) reflects the frequency of the term at *document-wise* level.

  $$IDF(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|} \tag{2.11}$$

  TF-IDF is not a single function. It is a family of weighting scheme in which there are slightly different members. For example, there are many variations of TF, such as *binary*, *raw frequency*, *log normalization*, and *min-max normalization*. Also, many variations of IDF are available. Frequently used variations are *IDF smooth*, *IDF max*, and *probabilistic IDF*.

Cosine is a measure of similarity between two vectors. It is defined as the inner product of the vectors divided by the product of their lengths.

$$\delta_{cosine}(X, Y) = \frac{X \cdot Y}{\|X\|\|Y|} = \frac{\sum_{i=1}^{n} X_i Y_i}{\sqrt{\sum_{i=1}^{n} X_i^2}\sqrt{\sum_{i=1}^{n} Y_i^2}} \tag{2.12}$$

In the context of string similarity, the vector X and Y are also represented by the *TF-IDF* weight of the tokens. Then, the inner product of X and Y is defined as the dot product of the weights of common elements between X and Y. Finally, the *TF-IDF cosine* similarity is written as follows.

$$\sigma_{cos}(A, B) = \frac{\sum_{t \in A \cap B} tfidf(t, A) \cdot tfidf(t, B)}{\sqrt{\sum_{t \in A} tfidf^2(t)}\sqrt{\sum_{t \in B} tfidf^2(t)}} \tag{2.13}$$

where $tfidf(t, I) = TF(t, I) \times IDF(t, D)$ and D is the document set where I belongs to.

In the context of instance matching, the instances are considered as documents and the terms are the tokens of string values. Then, the TF and IDF are computed similarly to those in the original problem. Also, I is replaced by the instance and D is its repository.

- **Okapi-BM25** (aka. BM25) is one of state-of-the-art corpus-based metrics and is also originally proposed for information retrieval problem [132]. Given a query Q, the $BM25$ of a document d is computed as follows.

$$\text{BM25}(d, Q) = \sum_{t \in Q} \text{IDF}(t, D) \cdot \frac{TF(t, d) \cdot (k + 1)}{TF(t, d) + k \cdot \left(1 - b + b \cdot \frac{|d|}{\text{avgdl}}\right)}, \tag{2.14}$$

where avgdl is the average document length in the corpus D. k and b are free parameters. The standard range of k and b are [1.2,2.0] and 0.75, respectively. In BM25, the IDF of the term t is usually the probabilistic IDF, which is computed as follows.

$$\text{IDF}(t, D) = \log \frac{|D| - n(t) + 0.5}{n(t) + 0.5} \tag{2.15}$$

where $n(t) = |\{d \in D : t \in d\}|$ is the number of documents containing the term t.

Similar to *TF-IDF cosine*, $BM25$ can be used in the context of instance matching, by replacing the query q and document d by two instances.

4. **Hybrid** The hybrid method combines metrics from different approaches in order to take the advantage of many metrics. The most common type is the combination of metrics on different level of inputs. Hybrid metrics usually take two levels. The metric for the first level is called primary and the other one is called secondary.

   For example, *Soft TF-IDF* [95] is the combination of *TF-IDF cosine* (primary) and a transformation-based metric (secondary) . *Soft TF-IDF* modifies the original *TF-IDF cosine* by replacing the intersection set $A \cap B$ in Equation 2.13 by

   $$T = \{t \in A | \exists t' \in B, \sigma(t, t') \leq \alpha\} \tag{2.16}$$

   where σ is a transformation-based metric and α is a threshold. The secondary metric σ can be Levenshtein, Jaro, Jaro-Winkler, etc...

   Another example is the *generalized Jaccard*. This metric applies $Jaccard$ on token sets but using $Jaro$ or $Levenshtein$ instead of exact matching. For example, by using $Jaro$, Equation 2.3 becomes:

   $$\sigma_{jaccard-jaro}(A, B) = \frac{|T|}{|A| + |B| - |T|} \tag{2.17}$$

   In this case, Equation 2.17 enables the approximate matching between tokens for an intersection-based metric.

2.3.1.2 URI

URI nowadays is a common data type and especially frequent in *linked data*. It is used to link an instance to the others having a relationship. In some cases, URI defines useful information for instance matching (e.g., *Wikipedia link*, *homepage*, and other functional properties).

Since URIs are stored as text, matching algorithms can use string similarity metrics for URIs. In this case, the domain part of the URI is usually striped if the URI contains a subdirectory. Otherwise, the whole original URI is used for matching. For matching URIs, exact matching is widely used as the precision of matching URIs should be at high expectation. Another option is dereferencing the URI to obtain further information, such as *page title* or *description*. To this aspect, dereferencing the linked data instances may bring plenty of useful information.

2.3.1.3 Numeric

Numeric is used to define the properties in *integer* and *real* ranges. Many important information are stored as number, such as *geographic coordinates*, *area total*, *population*, *people's height*, etc... There are some metrics to measure the similarity of numbers. Here we review two common metrics: *reverse difference* and *percentage difference*. We use A and B to denote the two input numbers.

1. **Reverse difference** measures the similarity of two numbers by taking the reverse of their absolute difference.

$$\sigma_{rdiff}(A, B) = \frac{1}{|A - B| + 1} \tag{2.18}$$

2. **Percentage difference** measures the similarity of two numbers by taking the complement of their absolute difference divided by the sum of them.

$$\sigma_{pdiff}(A, B) = 1 - \frac{2 \times |A - B|}{A + B} \tag{2.19}$$

   For percentage difference,the same difference produces smaller σ_{pdiff} when the input numbers become larger. That is the difference between this metric and the *reverse difference*, which ignores the magnitude of the input numbers.

2.3.1.4 Datetime

The last common data type is datetime. Similar to the metrics for URIs, the most common one for datetime is *exact matching* due to the importance of much datetime-related information (e.g., date of birth and establish). Another remarkable metric is *temporal inclusion*. This metric considers the input as numbers by unifying them into a relative representation, such as minutes or days from a pivot period. Then, the comparison is done by using numeric metrics.

2.3.2 Similarity aggregation

Similarity aggregation is the second step in matching score estimation. This step takes the literal similarities as input and produces the final matching score. Suppose the set of literal similarities is $\mathcal{L}$, the matching score is defined as $\text{score}(X, Y) = \omega(\mathcal{L})$, where ω is an aggregation function. There are various aggregation functions. The frequently used aggregations consist of *Minkowski*, *weighted average*, and *boolean*.

2.3.2.1 Minkowski distance

Minkowski distance is a metric in vector space. It originally measures the difference between two vectors. Let X and Y be the vectors of length n. The Minkowski distance of X and Y is defined as follows.

$$\sigma_{minkowski}(X, Y) = \left(\sum_{i=1}^{n} |x_i - y_i|^k \right)^{\frac{1}{k}} \tag{2.20}$$

where $k \geq 1$ is the parameter controlling the different magnitude of elements. Minkowski is the generalization of Manhattan and Euclidean distances. Indeed, when $k = 1$ and $k = 2$, Equation 2.20 becomes the distance of Manhattan and Euclidean, respectively.

By considering the literal similarities $\mathcal{L}$ as the differences of vector elements, the similarity aggregation based on Minkowski is equivalently defined as follows.

$$\omega_{minkowski}(\mathcal{L}) = \left(\sum_{\ell \in \mathcal{L}} |\ell|^k \right)^{\frac{1}{k}} \tag{2.21}$$

2.3.2.2 Weighted average

This aggregation takes the average of literal similarities and at the same time includes the expected impact of each similarity. The weighted average is widely used in instance matching because of its simplicity. This aggregation is defined as follows.

$$\omega_{w-avg}(\mathcal{L}) = \frac{1}{|\mathcal{L}|} \sum_{i=1}^{|\mathcal{L}|} \ell_i \cdot w_i \tag{2.22}$$

where $\mathcal{W} = \{w_i\}$ is a weight vector. When weight vector is a ones vector, the aggregation becomes the normal average of literal similarities.

2.3.2.3 Boolean aggregation

Boolean aggregation is not defined by a function but the transformation of original set of literal similarities. Using boolean aggregation is equivalent to applying a filter on $\mathcal{L}$ prior to an aggregation function, such as Minkowski or weighted average. That is, a filtered set of similarities $\mathcal{L}' = \{f(\ell_i) | \ell_i \in \mathcal{L}\}$ is extracted, where

$$f(\ell_i) = \begin{cases} 0 & \text{if } \ell_i < \alpha_i \\ \ell_i & \text{otherwise} \end{cases} \tag{2.23}$$

$\mathcal{A} = \{\alpha_i\}$ is a conditional vector. Each element of this vector defines a acceptance criterion for the respective literal similarity. Then, the matching score is calculated on the filtered set $\mathcal{L}'$.

Boolean aggregation offers the advantage of better discriminating the positive and negative co-references. By applying this strategy, the similarities of above threshold (i.e., α_i) contributes to the matching score whereas the under threshold ones make zero impact on the final result.

2.4 Determiner

The second component of instance matching is the determiner (Figure 2.1) . This component produces the final coreferences on the basis of matching score of instance pairs and sometimes the detailed literal similarities. In this section, we review some basic determiners, including three strategies of specification-based instance matching. In addition, we discuss the strategy using classifier as the determiner.

2.4.1 Top K selection

This determiner selects K instance pairs having the highest matching scores with respective to each instance of the source repository. Top K selection usually accompanies with a small threshold to avoid the acceptance of just slightly similar instances. The advantage of top K selection is the adaptive acceptance criterion, which is implicitly defined for each instance of the source repository. That is, the most similar instances are guaranteed to be selected if it satisfies a small threshold.

2.4.2 Thresholding

This determiner simply applies a global threshold for all instance pairs. Different from top K selection, this strategy may take highly similar pairs in the observation of all instance pairs. The instance pairs whose matching score is higher than the given threshold are selected for the final coreferences. The limitation of this strategy is that it may accept the pairs having a high matching score but non-coreferent. Such situations happen because of the inequality of ambiguity when observing instance pairs in the different group sharing different words. Matching instances containing frequently used words is more prone to ambiguity. However, the advantage of this determiner compared to top K selection is the ability to reject dissimilar pairs as it does not put an objective of K pairs for each instance of the source repository.

2.4.3 Classification-based

The determiner of this type does not work directly on matching score. It reuses the prediction of the classifier trained on curated data, which consists of some instance pairs and their matching label (positive or negative). The prediction given by the classifier on an unlabeled instance pair determines if it is included in the result or not. The classifier can solely take the final matching score or the literal similarities as the input. A matching system that applies classifier-based determiner is called a classifier-based matching system. This type of system does not require the matching specification because it can work independently with the instruction of property mappings, similarity metrics, aggregation strategy, and thresholds. All those instructions are optimized by the training process.

2.5 Evaluation metrics

The goal of instance matching is to return the instance pairs that are identical to the actual coreferences. Therefore, the evaluation metrics are conventionally designed to analyze the differences between these two sets. An instance matching system can be evaluated at different steps rather than only on the final results. In Figure 2.1, a basic matching system checks all pairwise alignments between given repositories. However, this practice is at the high complexity and time-consuming. The recent matching system applies blocking as a step that reduces the number of comparisons. That mission of blocking is to produce instance pairs that are less than the pairwise alignments but contains as many expect coreferences as possible. Therefore, the evaluation of blocking algorithms is as important as of the final result. Two evaluation metrics are used for blocking: pair completeness and reduction ratio. For the final result, which is the detected coreferences, there are three metrics: recall, precision, and F1. Let R_S and R_T are the source and target repositories, respectively; let I is the set of detect coreferences; A is the set of actual coreference; and C is the set of candidates (i.e., the result of blocking); the evaluation metrics are defined as follows.

- **Recall** reflects the ability of the system at recognizing as many actual coreferences as possible.

$$rec = \frac{|I \cap A|}{|A|} \tag{2.24}$$

- **Precision** reflects the ability of the system at discriminating the actual coreference and the impostors, i.e., the incorrectly detected coreferences.

$$prec = \frac{|I \cap A|}{|I|} \tag{2.25}$$

- **F1** is a conciliation of recall and precision. F1 is the harmonic mean of recall and precision.

$$f1 = 2 \times \left(\frac{1}{rec} + \frac{1}{prec}\right)^{-1} = 2 \times \frac{|I \cap A|}{|I| + |A|} \tag{2.26}$$

  It is possibility that a system obtains very recall but low precision and vice-versa. Therefore, F1 is considered to be the primary evaluation standard.

- **Pair completeness** reflects the ratio of correctly retained pairs after applying a blocking algorithm. The pair completeness is equivalent to recall but is computed based on the set of candidates.

$$pc = \frac{|C \cap A|}{|A|} \tag{2.27}$$

- **Reduction ratio** expresses the relative compactness of the candidate set compared to the pairwise alignments of instances.

$$rr = 1 - \frac{|C|}{|R_S| \times |R_T|} \tag{2.28}$$

All evaluation metrics have the range of $[0, 1]$ and the better performance comes with the higher value.

We reviewed the basic and commonly used techniques in instance matching systems. We also described how to evaluate the result of instance matching. Next, we discuss the remarkable work contributing to the problem of instance matching.

2.6 Related work

Instance matching was founded a long time ago and has been widely studied. It was first addressed by Halbert L. Dunn in early 1946 with the article titled "Record Linkage" [33]. The problem was to match persons using the events in their life. The foundation of modern *record linkage* is proposed in 1959 [96] and formalized after that ten years [39]. Since 1990s, the development of computer contributed to the promotion of research

on record linkage [160]. *Record linkage* has been studied under various names, including *entity resolution*, *coreference resolution*, *entity linking*, *data conciliation*, etc., and *instance matching*, the term we use in this dissertation.

There are some problems very similar to instance matching, such as *data deduplication*, *entity disambiguation*, and *ontology matching*. *Data deduplication* [36, 129] performs instance matching on one input repository. That is, the source and the target repositories are identical. *Entity disambiguation* [49, 53] matches the mentions in text and repositories' instances. *Ontology matching* [19, 34, 71] is the problem of mapping two ontologies, which include the instances of the classes, properties, and the relations (e.g., sub-classes and sub-relations). In instance matching, the mapping of properties is very important. Therefore, ontology matching is considered as a closely related problem to instance matching. In instance matching, especially for non-manual systems, the property mappings are detected by the heuristic or learning algorithm. In such a case, the alignment of property is almost identical to ontology matching in the aspect of schema mapping (not the class mapping). However, the main difference is that the property mappings used for instance matching is not necessary to be semantically correct. For example, in instance matching, 'first-name' could be match to 'full-name' if the full-name is unavailable for both of the repositories. But, in schema mapping of ontology matching, such mapping is considered as problematic. Furthermore, because instance matching has to deal with very large amounts of instances, the complexity of instance matching is very high compared to ontology matching. Instance matching can be combined with ontology matching. Such combination is potentially useful to improve the their accuracy. The detected coreferences can support the matching of properties and classes and vice-versa (e.g., two classes are same if many of their instances are coreferent). Combining the results of instance matching and ontology matching also brings further benefits such as the detection of instance type (e.g., Obama is a president). In this dissertation, we focus only on the problem of instance matching.

Researches on instance matching mainly include the proposals of *determiners*. However, because of the large number of studies and the relationship to other problems (e.g., *information retrieval*, *machine learning*, and *natural language processing*), the quantity of work contributing to other issues of instance matching are also considerable [36, 40, 44, 77, 129].

There are many classification criteria for the work related to instance matching. In this section, we first review the existing instance matching frameworks. After that, we review the previous work by each component of the general matching process. That is, if a study focuses on different components, it may be mentioned multiple times in this section.

2.6.1 Instance matching framework

Many instance matching frameworks have been presented [75]. Popular frameworks include SERF [10], MARLIN [15], Active Atlas [143], FEBRL [22], TAILOR [35], MOMA [152], SILK [155], LIMES [98], RIMOM [80], and AgreementMaker [27]. These frameworks can be divided into groups based on different criteria, such as input and supervision. Considering the input, the first six frameworks focus on relational data and the remaining work with linked data. Considering the supervision, Active Atlas, FEBRL, MARLIN, and TAILOR offer the mechanism of learning-based while the others focus on training-free, which is equivalent to specification-based matching.

- SILK [155] and LIMES [98] provides declarative languages for the user to specify the property mappings, similarity aggregation, thresholds, blocking strategy, and other minor parameters of the matching process. SILK and LIMES support a diversity of similarity measures and aggregation. The difference between SILK and LIMES is mainly about the efficiency. LIMES is a designed as a time-efficient framework and is empirically proved to be significantly faster than SILK. However, LIMES fails to response when being tested with large datasets [107]. Compared to LIMES, SILK is considered as an easy-to-use framework so that it is more widely used. SILK is implemented as identity resolution module of LDIF (Linked Data Integration Framework) [141].

- AgreementMaker [27], and RiMOM [80] are ontology matching frameworks that also support instance matching. These frameworks comprise an extensible architecture that enable performance tuning of a variety of matching methods (e.g., conceptual or structural granularity, manual or automatic). Although the main focus of RiMOM is ontology matching, this framework is among the state-of-the-art in terms of instance matching. The general idea of RiMOM is to combine different matchers to obtain the highest performance. AgreementMaker is also a famous framework in linked data community. This framework has been tested in practical applications and reported in the Ontology Alignment Evaluation Initiative (OAEI) competition [37]. Recently, AgreementMakerLight [38] is presented as a flexible framework extended from AgreemenMaker. AgreementMakerLight focuses more on efficiency as its target is large ontology matching problems. However, AgreementMakerLight currently supports only medium size datasets with thousands of ontology elements.

- Matcher combination strategy like RiMOM is also a widely supported in many frameworks. SERF [10] and MOMA [152] are also among them. SERF (Stanford Entity Resolution Framework) simply considers individual matchers as “black

boxes" and applies different algorithms to minimize the executions of each matcher by tracking the previously compared values. The combination of individual matchers is represented as a disjunction of matching rules. MOMA (Mapping-based Object Matching) provides an extensible library containing different matchers as well as combination operators (e.g., average, maximum, weighted average, and preference).

- MARLIN [15] and Active Atlas [143] are pure-supervised frameworks that follow the classification-based matching and provide the mechanism for training and prediction procedure. Active Atlas adopts semi-supervised learning for reducing the training examples. It combines several decision trees by voting strategy in order to pick the most informative examples for the next learning iteration. MARLIN (Multiply Adaptive Record Linkage with Induction) applies a two-steps learning. The first step is to optimize the matchers on a single attribute. The second step is to combine the tuned matchers using SVM algorithm. The main issue of MARLIN and Active Atlas is the scalability as their learning routine is expensive. Active Atlas follows semi-supervised that selects the next examples by checking all the training set. Meanwhile, MARLIN separates the optimizations into two steps which actually can be implicitly combined at once.

- TAILOR [35] and FEBRL [22] are hybrid toolboxes for instance matching supporting non-training as well as training-based matching. TAILOR provides five different similarity measures (e.g., hamming distance, edit distance, Jaro, and Soundex) and two rule-based matchers whose mechanism is the decision tree. FEBRL is developed for the biomedical domain and is a freely available as an open-source software. It supports a wide range of 26 similarity measures and applies SVM classifier.

The existing frameworks include several limitations. None of them support at the same time the extensible (e.g., the inclusion of similarity measure), flexible (e.g., the selection of determiner: specification or classification), and time efficient. Furthermore, memory efficient is not in their goals although some are designed with the mission of matching large datasets.

2.6.2 Similarity and matching score estimation

Similarity measure, mainly for strings, is an important component of instance matching. In addition to the basic similarities mentioned in Section 2.3.1.1, here we discuss other notably advanced similarities. We classify the similarities into three groups: corpus-based, similarity flooding, and learning-based.

- **Corpus-based.** Corpus includes the definition of raw text data and knowledge-base (e.g., WordNet). All knowledge-based measures focus on the semantic similarity of texts by leveraging the synonyms or hyponyms offered by the knowledge-base [93, 94, 124]. However, using synonyms in similarity measures is computationally expensive. For reducing the complexity, efficient methods have been proposed with respective to different targets. Instance matching and related problems are among the targets. Lu et at. propose an efficient algorithm for matching strings based on the synonym similarity [85]. A so-called selective-expansion, which utilizes a novel indexing structure SI-tree, has been presented. Rodríguez et al. propose a domain-independent semantic similarity measure for entity of different ontologies [133]. The general idea is to determine the similar entity classes by using matching methods over synonym sets, semantic neighborhoods and distinguishing features that are further classified into parts, functions, and attributes. STS [63] is similarity using a modified version of the LCS (Longest Common Subsequence). This measure uses both of corpus and lexical information in computing the similarity. In addition, different from other semantic measures, STS measure focuses on the similarity between two sentences or short texts.

  The most prominent statistic-based similarity is the Google similarity distance [25], which relies on a raw-text corpus. This similarity is estimated based on the co-occurrence probability of the given strings. Google similarity is used widely in many string matching tasks as it is believed to capture the semantic similarities such as synonym and hyponyms.

  Although semantic similarities are advanced measures, due to the complexity of this kind of semantic similarity, current matching systems have not included them. The main reason is that corpus-based similarity can be useful in instance matching when estimating the similarity of long strings rather than short strings. A short string usually contains the information of name and has less probability to be variant. However, a long string is frequently used to store a short description about the entity and have more chance to be modified with semantically related words.

- **Similarity flooding**. Traditional approaches compute the similarity of instances based on the literal similarities, which reflect the correlation of properties. Similarity flooding [87, 89] is an approach to estimate the similarity of instances when the literal similarities are unavailable but the relationship of instances are given (e.g. graph data). The general idea of similarity flooding is to consider the relationships as edges of a graph. The similarities of instances are propagated over the graph using structural information (e.g., degree of nodes and type of relations). Similarity flooding is commonly used in reasoning-based instance matching [87, 89].

- **Learning-based.** Learning of similarities is a considerable advantage of supervised instance matching. Many supervised systems treat the similarity learning as an important task [15, 81, 97, 143]. In such systems, a large collection of similarity measures is chosen at the beginning and refined during the learning process.

2.6.3 Property mapping

Property mapping is closely related to ontology matching [19, 71, 144]. However, ontology matching usually focuses on schema information (e.g., structural, lexical, and descriptive information) whereas property mapping in instance matching mainly is mainly the observation on instances.

Property mapping in instance matching is formed by the mechanism of determiner. For manual approach, which is schema-dependent, the mappings are constructed by user or domain expert [27, 48, 68, 69, 80, 80, 118, 125, 135]. For automatic approach, which is schema-independent, the mappings are automatically detected by statistical methods. These methods are mainly based on the overlap of values described by the properties. [5, 102, 112]. In addition, supervised specification-based instance matching is also considered as domain-independent because the property mappings are optimized by the learning process [58, 60, 61, 99–101]. For this approach, initial property mappings are explicitly or implicitly defined and a learning algorithm is used to select the optimal property mappings. Supervised methods so far produce the best quality mappings and sometimes even better than the manual approach. The reason is that they consider the relationship of properties based on the actual data, which a domain expert may not acknowledge by just investigating the schema and the experience on other datasets.

2.6.4 Blocking and indexing

2.6.4.1 Blocking

Blocking is to generate the candidates for instance matching by filtering only potentially coreferent pairs of instances. The early stage of blocking is naive approach, which applies an “accept all” filter on the pairwise instances [35, 50, 147]. This approach is computationally expensive and impractical for large repositories. Therefore, advanced blocking methods have been proposed [8, 23]. The notable methods comprise Key-based blocking [65], sorted neighborhood [50, 166], clustering [88, 121], weighting [62, 90, 98, 155], and learning [14, 29, 72, 73, 122].

- **Key-based** method [65] applies an index function for one or a few selected attributes and simply groups the instances that share the same value into one block.
- **Sorted neighborhood** [50] arranges the instances into an order of lexicographic feature and slides a fixed-size window to build blocks. Adaptive sorted neighborhood [166] is an advanced version of sorted neighborhood. This method instead of using a fixed size of sliding window adaptively adjusts the size based on the frequency of blocking keys. Therefore, it reduces number of candidates and improve the efficiency of the matching process.
- **Canopy blocking** is similar to iterative clustering. The method randomly selects a canopy center and the instances having acceptable distances to this center are put to current cluster. For reducing the size of clusters, a tight threshold is configured to determine the certain candidates.

  Baxter et al. conduct an interesting comparison between Key-based blocking, Sorted neighborhood, and Canopy clustering [8]. The experimental results show that Canopy clustering outperforms the two others when using the appropriate parameters.
- **Attribute clustering** is a clustering algorithm that groups the instances by similar values on given pair of properties. The most recent algorithm of this approach is the trigrams-based clustering [121]. It works by grouping properties having many overlaps between the values described by the properties. The overlap similarity is based on a trigram-based score. Each property pair in the same group generates a block of instances sharing a token described by those properties. Attribute clustering sometimes generates many candidates due to the rich ambiguity of some tokens. Block purging [120] provides a simple mechanism to prune off those situations. The technique is to discard blocks that have too many instance pairs. The general idea is to assume that a token shared by many instances makes less discrimination and tends to be a stop-word.

  Abovementioned methods consider only the sharing or non-sharing tokens. This approach offers the very high completeness of the blocking result. However, in some time-limited matching task, the number of candidates is still very high.
- **Weighting methods** [90, 98, 155] assign an impact factor for each indexed token and instance. Based on those factors, this method returns the candidates in accordance with a ranked list. Usually, weighting methods are used in hybrid with attribute clustering. For example, SILK [155] and Zhishi.Links [117] look up candidates using BM25 weighting scheme. SILK supports a user-specified interface to configure the mapping alignment, while Zhishi.Links uses the alignment for the name of instances.

On some clean datasets, a weighting method has proved to reduce zero loss in recall [62]. However, weighting methods require a threshold value of an expected K parameter to select the topmost related instances of the target repository for each instance of the source repository. Therefore, in many real large datasets, weighting method achieve a modest of 80% completeness. An example for this situation is using DBpedia spotlight on NYTimes-DBpedia dataset.

For non-training methods, the blocking keys and property mappings are selected by the domain expert and have shown generally effective performance [51, 161]. However, the manual process may be expensive, as the expertise may be unavailable for some domains. Therefore, learning methods are developed for reducing the human involvement.

- **Learning** approach iteratively optimizes the blocking model by reflecting the blocking result and expected candidates (supervised) or based on pseudo and heuristic metrics (unsupervised).

  Song and Heflin proposed an algorithm to combine multiple property mappings that increase the *discriminability* and the *coverage* metrics [148], which are assumed to express the goodness of the blocking model. Papadakis et al. introduce supervised meta-blocking that learns the classification models for recognizing the candidates [122]. The authors presented some generic features that combine a low extraction cost with high discriminatory power.

  Supervised blocking also attracts many researches. Bilenko et al. and Michelson & Knoblock focus on supervised adaptive blocking, which use learning algorithms for automatically choosing the instance's feature selection function [14]. BSL [92] employed supervised learning for blocking scheme, which is a disjunction of conjunctions of blocking functions. minHash [29] algorithm learns the blocking scheme for a given CNF determiner. This algorithm maximizes the pair completeness by generalizing the CNF clauses. Because deriving from the original CNF determiner, the model generated by minHash guarantees to produce the candidates with no loss in pair complemented compared to directly applying the determiner on pairwise instances.

  Recently, Kejriwal and Miranker propose a hybrid method that combines unsupervised and supervised learning for blocking model [72, 73]. The first step of this approach is to perform dataset mapping and the second step learns the blocking schemes on structurally heterogeneous repositories.

In addition, candidate generation is not only helpful for further matching, but also can be applied as a key component for interlinking. In [159], instance matching task is

conducted by combining a core entity resolution algorithm with the blocking and put them in an iterative process.

Although many blocking algorithms have been proposed, the scalability is still not fully solved as giving a billion scale datasets, state-of-the-art frameworks can not produce the candidate set with high completeness [98, 155].

2.6.4.2 Indexing

The inverted index is a widely used technique in many fields of data engineering. The index contains the terms as well as the respective documents in where each term appears. Inverted index is also used in instance matching to speeding up the lookup of instances given a value [2, 9, 23, 127, 156, 158, 164, 165].

The difference between systems or frameworks is the value representation (i.e., the original value or a coding of the value). For example, PPJoin+ [165] considers the order of tokens in an instance for the lighter index but faster retrieval. BiTrieJoin [156] supports efficient edit similarity with a technique called sub-trie pruning. Ed-Join [164] indexes the n-grams of tokens. IndexChunk [127] also used n-grams but also focuses on character-level to deal with the mismatching cases such as spelling errors. FastJoin [158] applies fuzzy matching that directly considers both token and character level similarity, rather than indexing the n-grams like IndexChunk and Ed-join.

Weighted index is an advanced technique that comprises both inverted index and the weight of terms and instances. DBpedia spotlight [90], SILK [155], LIMES [98], and [59] apply weighted index. The key technique is to index instances to enable efficient lookup with weight. Given a list of terms, the instances similar to the query can be efficiently retrieved. The retrieval results are ranked and presented with the relevant scores. Lucene[1] is a powerful weighted indexer. The main feature of Lucene is the fast text search, which is represented as a query on a database. Lucene is a commonly used indexer for information retrieval and implemented in many instance matching frameworks/systems, such as SILK, LIMES, Knofuss, etc...

2.6.5 Specification-based determiner

Specification-based determiners detect the coreferences using the specification of property mappings, similarity measures, and matching thresholds. There are many types of specification-based determiner and the classification criteria are also diverse. For example, if we consider the supervision aspect, there are unsupervised, semi-supervised,

[1] http://lucene.apache.org/

and supervised systems. If we consider the construction of specification, there are manual and automatic approaches. If we consider the usage of the specification, there are reasoning, crowdsourcing, and filtering-based systems. Because of that diversity, it is not completely separable if we divide the determiners by a particular criterion. Here we review the systems of notable groups instead of classifying all the previous work.

2.6.5.1 Manual approach

Manual approach constructs the specification with the help of the human, usually a domain expert. Given two repositories, an expert has to investigate the information of data domain, schema, properties, and sometimes with some example instances. Remarkable work of manual approach include RiMOM [80], LinkedMDB [48], MusicBrainz [130], Zhishi.Links [117, 118], KD2R [125], LN2R [135], LogMap [68, 69], and SAIM [86]. The mechanisms of these systems and frameworks are similar to the basic workflow of instance matching (Figure 2.1).

- RiMOM [80] works by estimating the similarity of each property mappings and combine them by a weighted vector. This vector is configured by a user and is tuned for different datasets.
- Zhishi.Links [117, 118] is among state-of-the-art matchers. This system performs two-step matching. In the first step, the label of objects are quickly compared and the very different instances are discarded. In the second step, a complex metric at a semantic level is applied. This system improves the efficiency by retrieving similar instances using a weighting scheme (e.g. TF-IDF, BM25). This mechanism is adopted from SILK framework. LinkedMDB [48] focuses on the movie domain. This system matches the films, directors, producers, and actors between the LinkedMDB database and the others (e.g., DBpedia, YAGO, Geonames, MusicBrainz). LogMap [68, 69] and MusicBrainz [130] also very similar to LinkedMDB and Zhishi.Links. The difference is that these systems focus only on matching the label of instances.
- RDF-AI [139] considers instances as a graph and uses graph matching algorithms. A graph of instances is represented by many nodes and edges, where nodes are instances and values, and the edges are the properties, and relationship between the instances.
- KD2R [125] is a key discovery algorithm that detects the properties with most distinct values. Although KD2R is an automatic algorithm, it relies on implicit

property mappings because it assumes the same properties are used for different repositories.

- SAIM [86] is a semi-automatic system which allows the user to define the property mappings. SAIM provides a simple and effective tool to support users in the creation of matching specifications.

2.6.5.2 Automatic approach

This approach constructs the specification automatically and is also called schema-independent because it works without the specification of the schema. SERIMI [4, 5], SLINT [113], and PARIS [151] are representatives of automatic approach.

SERIMI and SLINT select the useful properties and their alignments for the specification. SERIMI uses the entropy and SLINT uses the discriminability of the values. The similarity aggregation of these systems is the simple average. Although these systems adopt simple architecture, they have high capability of schema-independency and obtained good evaluation result. SERIMI placed at the second rank in the OAEI 2011 Instance Matching challenge [37]. Later, SLINT reported a similar experiment at a higher result. SLINT+ [102, 112] is the extension of SLINT with higher scalability but resolving the problem of expensive property mappings. However, SLINT+ cannot support large scale matching tasks because of the high computational operator in blocking step.

PARIS [151] is an automatic system which simultaneously matches the instances, classes, properties, and values using probability propagation. PARIS starts with the literal similarity on some conventional properties (e.g. name, label) and then estimates the equivalent properties, classes, and instances through many iterations. PARIS is considered as an advanced system because of the multiple goals. However, PARIS is not scalable because of the high computational complexity.

2.6.5.3 Reasoning

This approach applies logical expression or reasoning mechanism to determine the coreferent state.

- LN2R [135], CODI [119], and ASMOV [66] are the determiners which combine numerical and logical computation. The core idea of these systems is the disjunction of logical clauses which implement the numerical condition for each property mappings.

- Getoor pointed out that all instance matching tasks can be viewed as a clustering problem if the data contain the relationship between instances and thus can be represented by graphs [13]. Furthermore, for graph-based instance matching, the use of collective information is well-studied [11, 32, 84]. Notably, Dong et al. examined the relationships between instances on multiple aspects [32]. The relations are propagated over dependency graphs. The system was applied to multiple real world datasets and showed the good effectiveness. Melnik et al. and Marshall et al. proposed similarity flooding algorithms [87, 89] based on the reasoning over the graph-based repositories. The similarity of instances is indirectly computed using the structural information of the graph. The advantage of this approach is the independence on the property mappings as well as the literal similarities. However, the drawback is the potential weakness of applicability to relational data, on which the power of graph-based reasoning is not fully leveraged.

- Niepert et al. presented a probabilistic-logical framework for ontology matching [115]. This framework is based on Markov logic and defines the syntax formalization for ontology matching.

- Hogan et al. proposed an approach that combines reasoning and statistical analyzes [54]. This approach uses existing coreferences to predict the other relations. The semantics of functional properties, inverse-functional properties, and cardinality restrictions are applied.

- idMesh [28] lets users define the similarity between arbitrary instances then probabilistically infers the coreferences based on uncertain links using constraint satisfaction mechanisms.

- WebPIE inference engine [153] provides distributed computation over RDF graph. The objective of WebPIE is to scale up the reasoning engine into 100 billion triple tasks. In order to do that, WebPIE is built upon the Hadoop platform and is deployed on a computer cluster of 64 machines.

The main disadvantage of reasoning approaches is the heavy dependency on the correctness of logical expressions. The reasoning-based approach usually comes with high precision but low recall and should be used in matching task requiring high precision.

2.6.5.4 Crowdsourcing

Crowdsourcing for instance matching is a hybrid human-machine approach. The role of human is to verify the detect potential coreferences and the role of the machine is to

find and present those options. In instance matching, the machine usually determines the matching score based on machine learning approach or logical reasoning.

- Vesdapunt et al. proposed graph-based crowdsourcing framework for instance matching [154]. The input repositories are represented by graphs with all instances are nodes and the edges contain the probability reflecting the coreferent possibility (i.e., given by a machine Learning algorithm). The goal is to resolve the coreferences by asking users to verify the equality nodes. leveraging. Transitivity of the coreferent relation is also applied to speeding up the process.
- CrowdER [157],CrowdMap [137], and ZenCrowd [30] are also hybrid systems. CrowdER is instance matching system while CrowdMap [137] and ZenCrowd [30] are designed for ontology matching. However, the mechanism of these systems shares the general idea like [154]. The machine work also is the initial overall data and user are asked to verify the most likely coreferences.

2.6.5.5 Learning

Learning approach includes machine learning and supervised search algorithms.

- EUCLID [101], Knofuss [116], GenLink [60], and EAGLE [100] use genetic algorithm to learn the specification. The matching rule is encoded with property mappings and the goal of learning is to optimize the combination of mappings. Among these systems, only Knofuss is unsupervised and thus is equivalent to an automatic system. In such case, Knofuss considers pseudo-metrics for precision, recall, and F1 score as the optimization goal. The effectiveness that this system offers is not higher than SLINT+ and SERIMI but it takes a long time to produce a specification. This is also the main issue of the genetic algorithm-based system. Many iterations are needed to obtain the final optimal version of the specification.
- In order to speed up the learning process, ActiveGenLink [61] is designed as an active learning implementation of GenLink. Similarly, RAVEN [99] is also an active learning-based system that learns the specification that uses linear or boolean classifier as the determiner.
- ADL [58] is a recently proposed system for learning of specification. This system takes the input of two specific classes of instances and finds the good property mappings. The limitation of ADL is that it selects the mappings by considering each mapping independently with the others. That is, it ignores the collective

information of using multiple properties, which can lead to a redundant and bias determiner.

- ObjectCoref [55–57] employs a two steps approach for detecting coreferences. In the first step, an initial set of coreferences is obtained by using existing coreferences, inverse functional property, and functional property. In a second step, a machine learning technique learns the discriminability of property mappings. Based on that information, good property mappings are used to construct the determiner.

2.6.6 Classification-based determiner

Classification is applied in many problems. In instance matching, it is used to predict the coreferent state of two given instances. Recently survey and comparison between those systems can be found in [150] and [76].

- Bhattacharya et al. and Hall et. al studied the unsupervised instance matching and deduplication on bibliography domain [12, 47]. The authors argued that there should be dependencies between the collective information used for matching, such as title, author, and venue. The authors showed that Dirichlet process is useful for representing such dependencies and also can capture the relations between instance through a hidden group variable. The studies extended Dirichlet mixture by two different manners. In [47], a set of latent variables is utilized to monitor the Dirichlet-multinomial model for titles and a string similarity model for venues. In [12], Latent Dirichlet Allocation model is used to model the collaborative group to explain co-authorships.

  Unsupervised methods do not request training data. However, coming with this advantage is many drawbacks, including the complexity of learning algorithms and the low accuracy. Therefore, supervised instance matching is more popular.

- For supervised instance matching, many classifiers have been used. For example, SVM is used in MARLIN [15], FEBRL [22], and the work in [20]; Neural network is used in SEMINT [81]; Decision tree is used in LDIM [114], Semiboost [74], and the work in [134]. Using classifier, the systems need to construct the feature vectors, which is formed by the literal similarities and other useful information if any (e.g., class, type). Most systems follow the supervised mechanism and rely on curated data.

- In order to reduce the curation effort, semi-supervised, active learning and transfer learning are applied [21, 134, 138]. Semiboost [74] employs a semi-supervised learning approach that uses the trained model to predict the unlabeled data. The

examples with high confidence are used to retrain the classifier in order to improve the compatibility of the model and the data. This approach obtained good result compared to MARLIN and FEBRL but still far from perfect. Moreover, because the model is trained many times on high confidence examples, the initialization of the model is very important and is prone to over-fitting. Rong et at. applied active learning and transfer learning with decision tree classifier [134]. Transfer learning is done by manually mapping the properties of different repositories. The idea of transfer learning is important for extending the portability of supervised matchers.

The limitation of previous classification-based systems is that they simply employ the existing classifier, usually the state-of-the-art for instance matching. However, instance matching also has its special characteristics. One of the important points is that in the instance matching has to cope with ambiguity at the different levels for different terms (i.e. tokens, object values). Therefore, we are motivated from using supervised ranking in instance matching. The last section of our review is about the learning to rank techniques.

2.6.7 Learning to rank

Learning to rank is a subfield of machine learning that aims at predicting the order of data examples. The early form of learning to rank is the Ranking SVM, proposed in 2000 and re-exposition in 2002 by Joachims [70]. Ranking SVM is initially designed for optimizing search engines. In [70], the author constructed the training data by click-through logs (i.e. helpful search results according to users' preference). Each example reflects the relation between the query and the opened web pages in the order to clicks. That is, given a query and some web pages, the trained model can be used to rank the web pages by their relevance to the query.

In general, learning to rank can be applied to any problem whose performance is supported by a correctly ranked list of relevant examples. Learning to rank has not been used in instance matching. In this dissertation, we argue that instance matching can also be viewed as an information retrieval problem where the query is the instances of source repository and the task is to retrieve the most relevant instances in the target.

After the introduction of Ranking SVM, many approaches for learning to rank have been presented. [18]. Liu [83] categorized learning to rank algorithms into three groups: Pointwise [26, 42, 142], Pairwise [17, 41, 70, 142], and Listwise [17, 163].

- **Pointwise:** Each example represents a pair of entities and has a numerical or ordinal score. That is, the input of pointwise approach is similar to that of usual regression problem. Fuhr applied polynomial regression [42], Cooper et al. and Sculley applied logistic regression [142] for information retrieval problem.

- **Pairwise.** This is an advanced version of pointwise approach. In this case, the ranking problem is formulated by a classification problem. Each training example is an ordinal combination of two entities and the accompanied label for training is the binary value reflecting the correctness of order. For example, given for a query q and three documents d_1, d_2, d_3. For three documents, there are six pairs $(d_1, d_2), (d_2, d_1)$, (d_1, d_3), (d_2, d_3), and (d_3, d_2). The label for each pair is either 1 or -1 (or 0) which denotes the order is correct or not.

  Ranking SVM [70] is the first and the most widely used among pairwise learning techniques. Rankboost [41] combines pairwise ranking and boosting approach to of produce highly accurate prediction rules. CRR [142] linearly combines logistic regression and pairwise ranking. The experiment result showed good performance on LETOR [128], a standard dataset of learning to rank. However, CRR is a shortage of an interpretation of the probabilistic model.

- **Listwise.** Cao et al. argued that pairwise ranking ignores the fact that the final goal of ranking is to build a ranked list, not to predict only the order of two entities [17]. The authors proposed the first framework for listwise ranking that uses Neural network on the top of probabilistic models, which represent the relation between entities in a given list. Listwise is considered as the most difficult learning to rank task because it has to model the order of a list into an example. The optimization process is to minimize a loss function defined on the predicted list and the actual data. Xia et al. conducted a theoretical analysis of listwise ranking based on the loss functions, which directly affect the consistency, soundness, continuity, differentiability, convexity, and efficiency [163]. Based on that analysis, the authors proposed ListMLE, which is proved to be better satisfy those properties compared to other listwise approaches.

Except pointwise ranking, the other two learning to rank approaches usually have to cope with large training data because of many combinations of entities compared to the traditional learning task. Therefore, an efficient method for training the model is important. The model of learning to rank can be optimized by gradient descent [16] and thus the efficient stochastic gradient descent [142].

We have reviewed notable work related to instance matching. Next, we summarize the limitations and the spaces of improvement.

2.7 Open problems

From the review of related work, we can find that instance matching is a well-studied problem that attracts many researches. Nevertheless, there are still many unresolved problems. Among them, the prominent problems include the following points.

- **The shortage of a highly scalable framework:** Although there are some frameworks for instance matching, they are not really fit to the up-to-date big data. Matching on large repositories becomes difficult task for existing framework because they were built upon complicated data structures designed for multiple purposes. In addition, the mechanism of those frameworks ignores the issue of workload and memory balancing. That leads to the intractable memory management and fails to work with large repositories.

- **An automatic schema-independent approach:** In the context of unsupervised, when the curated data for automatic construction of matching specification is not available, the manual approach has shown the acceptable results. However, being able to work with any repository is very important for an instance matching system and manual approach fails to satisfy this requirement. Given a repository without any description of the schema or belonging to an inexperienced domain, it is difficult to accurately define an effective specification. The schema-independent approach is then very important to be studied.

- **More effective and efficient blocking scheme:** With the desire of high pair completeness (Section 2.5), many methods considered character-level of string tokens and produce too many candidates. On the other hands, some methods applied weighting to reduce the size of candidate set but failed to retain an expected number of coreferences. Weighting methods also suffer from high complexity and is not suitable for large datasets. Besides that, previous blocking schemes were mainly tested small, artificial, or limited domains of data, so that the performance on real and large data is not fully evaluated. A study on simple but effective blocking strategy and evaluation on plenty of domains are therefore very important. Furthermore, there is a necessity of developing a learning algorithm for blocking scheme in the supervised scenario.

- **Better learning algorithm for supervised specification-based matching:** The genetic algorithm so far is the best specification learning option in term of effectiveness [60, 100, 116]. However, it is empirically inefficient because of taking too many iterations to converge. ADL [58] offers a minimal cost algorithm but fails to find the optimal specification because it ignores the effect made by the

combination of different settings. More effective and efficient learning algorithm is considered as one of the unresolved problems.

- **Further improvements for classifier-based matching:** Current classifier-based matching systems do not include the mechanism of adaptive filtering, such as the effective stable matching. The stable matching can also be interpreted as the local ranking of an instance pair in its block. However, the objective of ranking is not to discriminate the positive and negative of coreferences. That is, using either ranking or traditional classification is insufficient to deliver the optimal performance. Therefore, the combination of ranking and classification, which makes the classifiers aware of the ranking factor is a feasible but unsolved task.

2.8 Summary

Instance matching is an important problem with many challenging issues. The issues cover from heterogeneity, ambiguity, and scalability to the detail techniques, including property mapping, blocking, similarity estimation, and the determiner.

In this chapter, we described the basis of the techniques used in instance matching. We also reviewed the related work in order to address the unresolved problems. Such problems consist of the shortage of a scalable instance matching framework, the necessity of schema-independent approach, more effective blocking, the supervised methods for specification construction, and ranking method for classifier-based systems.

On the basis of those limitations, we envision the space for improvements and the motivations of our work. As we mentioned in Section 1.2, we have achieved some optimistic results. From Chapter 3 through 6, we describe the detail of our contributions.

Chapter 3

ScSLINT: Framework for large scale instance matching

This chapter describes the scalable instance matching framework *ScSLINT*. We begin with the motivation of developing *ScSLINT* (Section 3.1). After that, we detail the framework (Section 3.2). In experiment (Section 3.3), we show that *ScSLINT* is efficient in term of time and memory. The comparison to other frameworks is also reported.

3.1 Motivation

The development of digital data induces the scalability issue to all problems. Instance matching is not an exception. Existing frameworks for instance matching have shown the usefulness. However, still there are some limitations, not only scalability but also other issues.

- **Indexing.** Previous frameworks used either weighted index or inverted index. The weighted index has the advantage of retrieving the top potential coreferences but the drawback is the low pair completeness (Section 2.5). The inverted index is fast but none of the existing frameworks using this index structure is freely available. Furthermore, existing frameworks reuse the indexing modules whose job is to manage many indexes different from the need for instance matching. Therefore, the index is complicated and may not reach the optimal performance. We focus on building a framework which follows the inverted index and is an open-source software so that it may be useful for further researches.

- **Workload.** The balance between memory and time efficiency is very important. Existing frameworks ignore the management of workload. They either incrementally use the repository (i.e. process one instance at once) or load the bulk of the whole repository into memory [98, 155]. Processing the data by this manner may fail to deal with scalability on large datasets. We take the problem of workload balancing into account when developing the framework.

- **Extensibility.** A framework may not support all functions such as different data acquisitions, similarity measures, blocking strategies, or determiners. Therefore, extensibility is important for users who expect a modification on existing framework. As previous frameworks do not allow such extensions, it is impossible to include a new algorithm or any similarity metric. We aim at developing a framework that supports the injection of new elements in every module of the matching process.

- **Portability.** Different from previous frameworks, we treat the input repositories as a unified representation, in which the instances are stored as a list of property-value pairs. An indexing module can be used to build the same data format from different inputs. Therefore, the framework in our objective is to work with different types of data (e.g., relational data and linked data).

- **Soundness.** Some of previous frameworks [98, 155] are built for large scale instance matching. However, they so far satisfied only the matching tasks on small

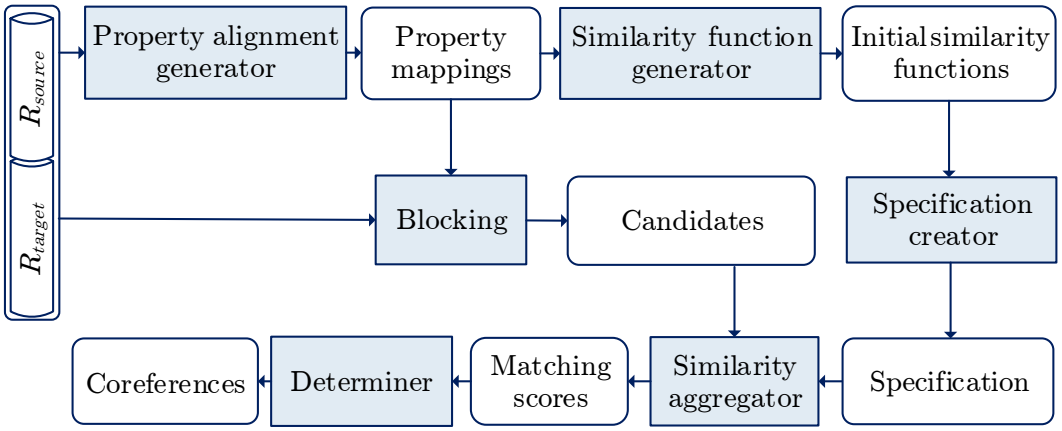


FIGURE 3.1: The architecture of *ScSLINT*.

and medium datasets, according to the reports. We are interested in testing the frameworks on very large datasets to see the capability.

We developed *ScSLINT* framework [107] for instance matching on large scale datasets. The framework provides the workload balancing mechanism, extensibility, portability, and the performance is verified on very large datasets. *ScSLINT* is written in C++ language and is optimized for Linux and Windows systems. The binary and source code of *ScSLINT* are openly available at `http://ri-www.nii.ac.jp/ScSLINT/`.

3.2 Methodology

3.2.1 Workflow

The architecture of *ScSLINT* is depicted by Figure 3.1. The input R_S and R_T are the indexed repositories. The output is the set of coreferences. In order to produce the output, there are six components need to be executed: property alignment generator, similarity function generator, blocking, specification creator, similarity aggregator, and determiner. The mission of each component is as follows.

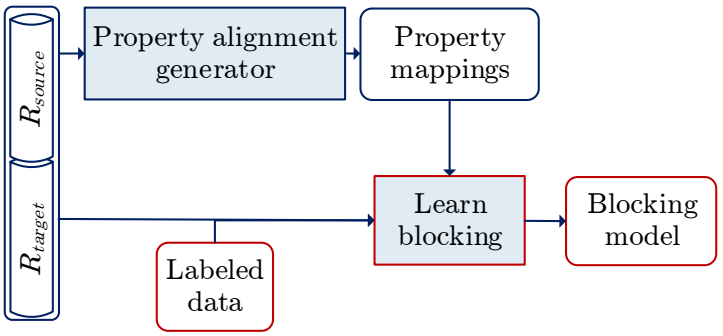
- **Property alignment generator** produces the property mappings between the schemas of R_S and R_T. Each mapping is expected to describe the same attribute of the instances
- **Similarity function generator** creates the similarity functions based on the generated property mappings.

- **Blocking** creates the candidates for matching. A candidate is a pair of instances having potentiality to be coreferent.
- **Specification creator** generates the specification of matching score estimation. A specification includes the similarity functions, similarity aggregation function, and necessary thresholds. For classification-based matching, the specification is used only for computing the literal similarities. In that case, the similarity aggregation function and other matching threshold are ignored.
- **Similarity aggregator** executes the similarity functions for each candidate. For a specification-based matching, this component computes the final matching scores and feed those results to the next step. For a classification-based matching, this component delivers all literal similarities estimated by similarity functions.
- **Determiner** produces the coreferences. For a specification-based matching, it relies on the specification and the matching scores. For a classification-based matching, the determiner is a classifier and the input that it takes are the literal similarities.

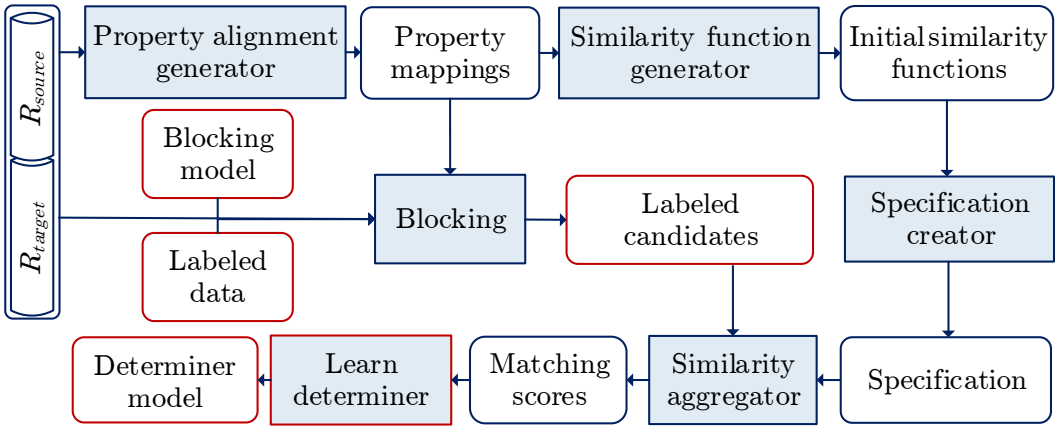
The above workflow is the general architecture of *ScSLINT*. Each component allows user's intervention to define the processing mechanism. For supervised instance matching, the workflow can be applied for labeled data and unlabeled data separately. Concretely, the illustration of workflow for supervised scenario is given in Figure 3.2. According to this figure, the supervised instance matching consists of three steps. The first step is the learning of blocking model. In this step, labeled data, which is the set of known coreferences, are input to the learning algorithm, in order to generate the blocking model. In the second and third steps, blocking model is used to generate the labeled candidates and unlabeled candidates, respectively. Labeled candidates are used to learn the determiner model. The last step is to apply the learned determiner on the candidates generated by the trained blocking model.

The learning of blocking model and the generation of candidates can be considered as equivalent to the blocking component in the general workflow. The determiner model also plays the role of the specification. Therefore, *ScSLINT* is generic and is compatible with multiple approaches. In addition, in each component of *ScSLINT*, we implement the support tool for the mission of the component, rather than the algorithm solving the mission. The extensibility of *ScSLINT* is reflected at the point that it allows the intervention at any component by including the algorithms (e.g., property mapping algorithm, and blocking algorithm) for the components.

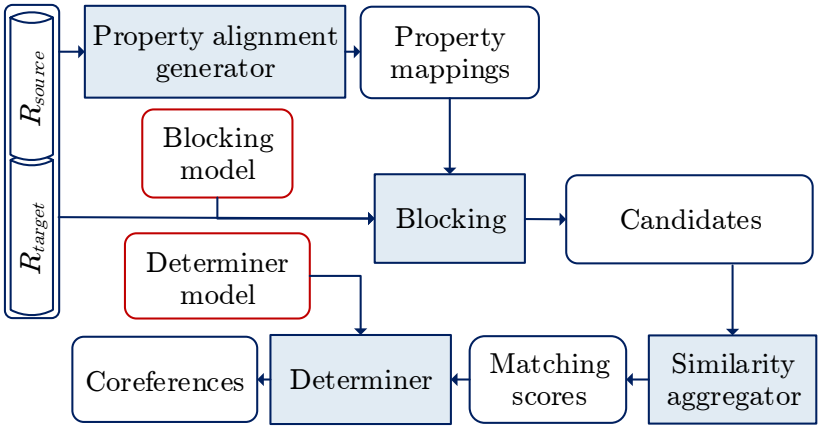
Next, we describe the detail of each component.



Step 1. Learning of blocking model



Step 2. Learning of determiner model



Step 3. Coreference resolving

FIGURE 3.2: Supervised instance matching with *ScSLINT*.

Table 3.1: Example of property mappings and assigned similarity measures.

Source property	Target property	Similarity measures
label	name	Levenshtein, TFIDF-Cosine
	alias	Levenshtein, TFIDF-Cosine
	leader	Levenshtein, TFIDF-Cosine
establish	named date	ExactMatch
population	area size	Reversed difference
	elevation	Reversed difference

3.2.2 Property alignment generator

This component creates the property mappings between the schema of R_S and R_T. A property alignment is expected to describe the same attribute. In order to detect such mappings, an algorithm is used to compare the values described by two properties. Therefore, *ScSLINT* provides the tool for fast querying the values. The queries are done upon an indexed structure, which is described in Section 3.2.8. Furthermore, for very large datasets, a frequent property may describe the values whose quantity is almost the size of the repository (e.g., name and label). In this case, comparing just two properties may be a scalability issue to many machine. *ScSLINT* supports partial querying for each property for solving this issue.

3.2.3 Similarity function generator

This component uses the property mappings generated previously to create a list of initial similarity functions. A similarity function is specified by two pieces of information: a property alignment $\langle p_S, p_T \rangle$ and a similarity measure σ. For two instances $x \in R_S$ and $y \in R_T$, a similarity function calculates the similarity $\sigma(p_S(x), p_T(y))$ where $p(x)$ extracts all values of x declared by p. Examples of property mappings and similarity measures are provided by Table 3.1.

ScSLINT supports some built-in similarity metrics and is extensible for any new metrics. In *ScSLINT*, the data type of property is taken into account. That is, an intervention can specify the suitable similarity metrics for different data types. The current version of *ScSLINT* includes the following metrics. For *decimal* values, *ScSLINT* supports the reverse difference (Section 2.3.1.3). For *date* and *URI*, *ScSLINT* supports exact matching. For *string* values, *ScSLINT* supports TF-IDF, Levenshtein, Jaro, Jaro-Winkler, exact matching, Jaccard, Dice, and common substring.

The similarity function generator creates only the prototype rather than execute (i.e. invoke) the functions. The similarity functions are later executed in similarity aggregator component.

3.2.4 Blocking

The mission of the candidate generator is to reduce the huge number of pairwise alignments between instances from $|R_S| \times |R_T|$ pairs into a significant smaller number. This component detects the candidates of potentially coreferent instances. In order to obtain that goal, the basic idea is to group the instance sharing some similar tokens. Therefore, *ScSLINT* provides the mechanism of fast retrieval of the instances containing the interested tokens. This is done by the index structure, which will be described in Section 3.2.8.

3.2.5 Specification creator

A specification contains the parameters for further components. It describes similarity functions, similarity aggregator, and determiner. This component is installed to wrap the further extension in instance matching. For manual instance matching, specification is created by user or expert. For automatic instance matching, specification is created by a heuristic method that constructs the similarity functions, similarity aggregator, and the parameters of the determiner. For supervised instance matching, this component plays the role of initial specification creator. The initial here is a default specification that is later refined by a learning algorithm. For classification-based classifier, the default specification is important to create the examples, which are the vector of literal similarities.

3.2.6 Similarity aggregator

In this component, the similarity functions are executed and their results are then accumulated into one final matching score. This is the most expensive component in the matching process. The input of this component are the candidates. For large datasets, the number of candidates can reach to billion pairs and storing all instances in main memory is very costly. *ScSLINT* provides the mechanism of incremental processing in this step. The detail is described later in Section 3.2.9. *ScSLINT* also applies parallel processing for performance multiple comparisons at the same time.

For aggregator function, *ScSLINT* currently supports the following combinations: sum, maximum, average, euclidean distance, and boolean aggregation. For boolean aggregation, the threshold for each literal similarity need to be specified by the intervention.

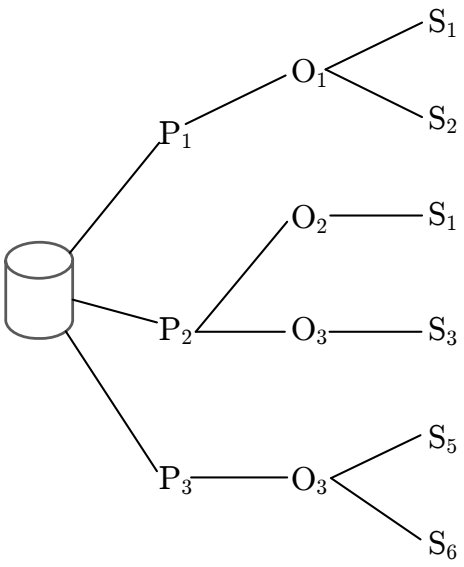


FIGURE 3.3: P-O-S indexing.

3.2.7 Determiner

ScSLINT support different types of determiner: specification-based and classification-based. A specification-based determiner usually applies thresholding or filtering techniques. A classification-based determiner reuses classifier model to predict the coreferences. Therefore, in this component, all *ScSLINT* need to provide is the information access on the matching scores and the detailed literal similarities of candidates. Based on such information, any determiner can find the necessary data for producing the coreferences.

We described the components of *ScSLINT*. Next we describe how *ScSLINT* is scalable by indexing and workload balancing techniques.

3.2.8 Indexing

The purpose of indexing is to provide a fast access to the repositories on any detail that is useful for the instance matching process. The first component of *ScSLINT*, property alignment, requires the index pairs of <property, token> for fast measuring the similarity of properties. The blocking component requires the nested index pairs of <<property,token>, instance>. Therefore, the first inverted index structure is designed for these two purposes. An example of this index is given in Figure 3.3. In this figure, S,P, and O stand for instance, property, and token, respectively. A token is defined as a word of string or the whole object value of other data types.

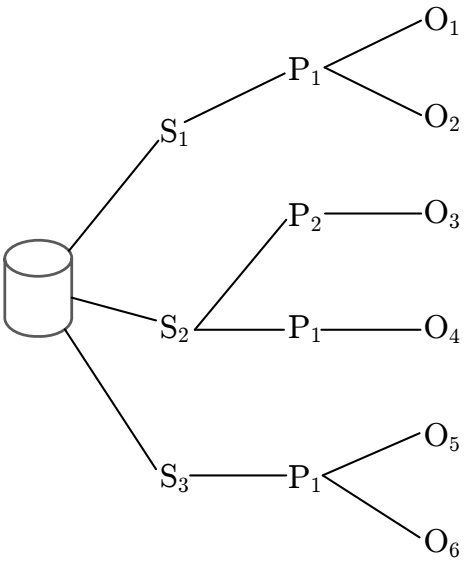


FIGURE 3.4: S-P-O indexing.

The second index structure is built for the similarity aggregator. This component takes two instances as input, estimates the literal similarities, and aggregates the matching score. Therefore, an index of <instance,property>,value> is necessary. An example of this index is illustrated in Figure 3.4.

All index structures provide an O(1) complexity for accessing an expected key-value. Therefore, the information access in *ScSLINT* is considered to be at maximum performance. The building of these indexes is very fast, at a linear complexity, with respective to the input repository. Furthermore, because the index provides random access to any indexed key-value pair, the workload balancing becomes possible with simple techniques.

3.2.9 Workload balancing

Workload balancing is the technique of using a limited amount of main memory for computation, given a large data that cannot fit all-at-once in the memory. This issue frequently happens for many large scale computational tasks. Instance matching is not an exception.

During the matching process (Figure 3.1), almost all components of *ScSLINT* have to cope the problem of scalability. The first step is the matching of properties. A repository can contains millions of properties in its schema (e.g., Freebase). Fortunately, the comparison between two properties can be done by querying only their information, instead of loading all properties. However, given a very large repository, a frequent

property may describe million values (e.g., name, label, and description) and thus billion tokens can be indexed. Therefore, even for one property, the workload management need to be taken into account. In order to speed up this step, *ScSLINT* offers the mechanism of partially querying the tokens of a given property.

In similarity aggregator, one important task is to estimate the literal similarities, given many pairs of instances. It is obvious that loading a bulk of instances at one time is more efficient than fetching one-by-one. However, an instance may be used multiple time if it involves with different candidates. In that case, loading an instance many times reduces the efficiency. Therefore a caching technique and a sorted set of candidates are utilized. Caching is to keep the frequently used instance in the main memory. The candidates are sorted by the order of a repository, usually the target repository as its has larger size in most matching tasks. The ordering of candidates enables more effective caching. An instance is kept in the main memory continuously for all comparisons related to itself. An instance is removed from the cache only when it is not needed anymore. In other words, an instance is loaded only one time for the repository selected for caching.

In fact, the above techniques are simple, but existing frameworks have not supported yet, although it is possible to be included [155]. Those frameworks load all properties or instances into main memory or storing all information on external memory. The former causes the problem of memory shortage and the latter is a significant reduction of processing speed. *ScSLINT* delivers a balance trade-off to obtain better performance on a limited memory machine.

3.2.10 Use-cases

ScSLINT is developed in 2014 and is published in 2015 [107]. The use-cases of *ScSLINT* so far include *ASL* [108], *ScLink* [111], and the learning of *R2M* feature.

- *ASL* is an automatic system focusing on domain-independent instance matching.
- *ScLink* is a supervised system focusing on high accuracy matching for heterogeneous repositories. *ScLink* is the result of a well-studied progress, whose products include different techniques presented in different venues [104–106, 109].
- *R2M* is a feature that aims at combining the learning to rank and instance matching, in the context of classification-based instance matching.

The detail of *ASL*, *ScLink*, and *R2M* are described in Chapter 4, 5, and 6, respectively.

Table 3.2: Datasets used for testing *ScSLINT*.

ID	Source repository			Target repository			Size
	Name	#Instances	#Properties	Name	#Instances	#Properties	($\times 10^9$)
D1	NYTimes loc	3,840	22	DBpedia	4,183,461	45,858	16.06
D2	NYTimes org	6,045	20	DBpedia	4,183,461	45,858	25.29
D3	NYTimes peo	9,958	20	DBpedia	4,183,461	45,858	41.66
D4	NYTimes loc	3,840	22	Freebase	40,358,162	2,455,627	154.98
D5	NYTimes org	6,045	20	Freebase	40,358,162	2,455,627	243.97
D6	NYTimes peo	9,958	20	Freebase	40,358,162	2,455,627	401.89
D7	NYTimes loc	3,840	22	Geonames	8,514,201	14	32.69
D8	DBpedia	4,183,461	45,858	Freebase	40,358,162	2,455,627	168836.8

3.3 Experiment

3.3.1 Experiment target

We evaluate *ScSLINT* in the aspect of memory and time efficiency. For the first issue, we simply deploy *ScSLINT* on a desktop computer with a usual computational power. For the second issue, we measure the time for building the repositories' index, and the detailed running time of each component. We also compare the runtime of *ScSLINT* and other state-of-the-art frameworks, including SILK [155] and LIMES [98]. Very large datasets are used for this experiment in order to verify the scalability practically.

3.3.2 Datasets

We use seven standard datasets selected for OAEI 2011 instance matching challenge. The repositories related to these datasets are NYTimes[1], DBpedia[2], Geonames[3], and Freebase[4]. The source repository is NYTimes and the targets are the others. For these datasets, NYTimes is divided into three subsets regarding to three data domains: location (loc), organization (org), and people (peo). For DBpedia and Freebase, every data domain of NYTimes form a dataset. For Geonames, only one dataset for location is created. Therefore, in total there are seven datasets. We also use another dataset connecting DBpedia and Freebase. The summary of all datasets are provided in Table 3.2. The only criterion for selecting the datasets is the scale. The first seven datasets are billion scale and the last dataset is trillion scale.

Some of previous systems reported experiments on the first seven datasets. However, they utilized simplified version of the datasets rather than the original data. Concretely,

[1]NYTimes version 2014/02
[2]DBpedia version 3.9
[3]Geonames version 2014/02
[4]Freebase version 2013/09/03

for DBpedia and Freebase, only the classes related to the subsets of NYTimes are used. They are people, location, and organization classes of DBpedia and Freebase. In fact, the ground truth of these datasets show that not only the instances in the appropriate class are the expected coreferences. That is, other experiment use a simplified version of the data for reducing the complexity, although the recall definitely drops. Some other experiments utilized a more simple datasets, in which only the coreferent instances are stored in the input.

The size of input repository is not only the matter of scalability. As we discussed in Section 1.1, the scalability also brings the problem of heterogeneity and ambiguity. However, in our experiment, we focus on the scalability of the frameworks.

3.3.3 Experiment settings

For using *ScSLINT*, we install the following configuration for its components.

- **Property alignment.** We do not map the property manually in order to make *ScSLINT* performe the property comparison task. We define using Jaccard index (Section 2.3.1.1) for this task. Two properties are considered as equivalent if the Jaccard similarity of them is higher than 0.75.
- **Blocking.** We apply a simple token-based blocking techniques. We also use the result of property mappings for this component. Given the list of property mappings M, two instance are considered as a candidate if they share at least the first token of the values described by any mapping in M.
- **Similarity function generator.** We apply two complex similarity measures for string type, Levenshtein and TFIDF cosine. That is, for each string mappings, two similarity functions are created. For other data types, we simply use only the exact matching.
- **Specification creator.** We use specification-based instance matching for this experiment. That is, we need to declare the thresholds for determiner and similarity aggregator. However, because we do not focus on testing the accuracy, we only declare the similarity aggregator. We choose average aggregation as the default for the experiment. The two last components work on the basic of this specification.

The computer used in the experiment is equipped with one Intel core i7 4770K CPU and 16GB memory. The CPU is a quad-cores architecture with virtual dual-core technology for each physical core. Therefore, we enable multi-threading with 8 threads for the similarity aggregator component.

3.3.4 Results

We report the runtime result of *ScSLINT* on two aspects: the time for building the indexes and for the whole matching process.

3.3.4.1 Indexing

The result of runtime for building the indexes is as follows.

- **NYTimes location:** 0.069 second.
- **NYTimes organization:** 1.006 seconds.
- **NYTimes people:** 1.908 seconds.
- **Geonames:** 1,362 seconds.
- **DBpedia:** 2,874 seconds.
- **Freebase:** 25,200 seconds.

In general, the time for building the index is very fast and is linear to the size of the repository. *ScSLINT* creates the index for Freebase in one hour on a usual desktop computer. A worthy note is that the size of Freebase is 198GB and the indexes cost in total 74GB. That is, 66% of the original data has been compressed using the index structures. It can be said that *ScSLINT* offers fast index that is sufficient for instance matching tasks and compact in term of size.

3.3.4.2 Matching process

The detailed runtime of the matching process is reported in Table 3.3. In this table, the second and the third column display the number of candidates and similarity functions. These two factors are the source of complexity for other similarity aggregator component. The other columns are the runtime of the respective components. According to this table, the most expensive component is similarity aggregator because it has to execute many string similarity measures. That is, the role of candidate generator is very important for reducing the number of candidates.

The longest matching time for NYTimes data is 36 minutes, on D7, which requires 11.16×10^9 comparisons. On the simplified version of this dataset, which is 2000 times smaller, the required time for LIMES and SILK, which are among the state-of-the-art

Table 3.3: Runtime of *ScSLINT*. (Unit: second)

Dataset	#Candidates ($\times 10^6$)	#Similarity functions	Property alignment	Blocking	Similarity aggregator
D1	32.2	12	37	7	70
D2	38.2	25	43	8	268
D3	61.7	17	46	11	404
D4	46.9	22	46	11	251
D5	222.7	23	14	111	641
D6	357.4	16	14	268	1023
D7	620.1	18	15	507	1578
D8	45,286.4	12	52	11,807	345,495

frameworks, is almost 30 minutes and 9.5 hours, respectively [98]. Because there was no report of an experiment on a billion scale dataset like D1 to D7, we tried to test SILK [155] and LIMES [98] frameworks on these datasets. The result is that LIMES and SILK fail to finish the matching task even within 10 times longer period for each dataset. In such cases, we terminate the processes after such period.

The result of *ScSLINT* on D8 is impressive. *ScSLINT* can finish this task and therefore it is not only the first framework tested on billion scale datasets (D1 to D7), but also the first framework successfully performed the trillion scale matching task. The total time to finish the matching on D8 is 99 hours. *ScSLINT* can process huge data like D8 using a usual computer. That is, the framework expresses its high scalability and can far benefit from being deployed with other big data processing framework or on a powerful machine.

3.4 Summary

Instance matching framework plays an important role in all matching tasks. Because existing frameworks do not fully support large scale instance matching, we are motivated from building a more scalable framework. *ScSLINT* is the result of that motivation. *ScSLINT* also provide more extensibility, portability, and memory efficiency. Furthermore, the performance of *ScSLINT* is verified on very large datasets that other frameworks cannot obtain the same results. This chapter described the detail of *ScSLINT* and experiments confirming the performance of the framework. The fact that *ScSLINT* enables the scalable instance matching is not simply an achievement of a powerful framework. It is also the start of further algorithms and systems whose objectives include the large matching tasks. Based on *ScSLINT*, we develop other systems named *ASL* and *ScLink*. Chapter 4 and 5 are the detail of those systems.

Chapter 4

ASL: Schema-independent specification-based instance matching

This chapter describes *ASL*, a schema-independent specification-based instance matching system. We begin this chapter with the motivation (Section 4.1) and the problem statement (Section 4.2). After that, we describe the detail of the system (Section 4.3) and report the experiment (Section 4.4).

4.1 Motivation

Heterogeneity and scalability are two of the major challenges of instance matching (Section 1.1). Heterogeneity includes the difference in data representation of real-world objects (e.g., Tokyo and capital of Japan) and the inconsistency of schemas, in which the attributes of objects are declared through arbitrary properties (e.g., 'name' and 'label' co-describe the same attribute). Scalability is mainly about the number of properties and instances of the input repositories. If a large number of properties is difficult for property mapping, that of instances directly contributes to the complexity of instance comparison.

Bypassing the inconsistency of schemas is very important because it determines the applicability of a matching system. In order to bypass this challenge, numerous studies have been published. Previous approaches can be categorized into manually operated, semi-automatic, and automatic systems. Using manual systems [68, 80, 155], user needs to provide matching specifications (e.g., property mapping and similarity measures). Semi-automatic systems try to reduce the user involvement by suggesting a specification [86] or by requiring a small number of labeled data [61, 99]. Recently, studies on automatic approaches have increased because of their generality. Existing automatic systems can be categorized into three families: unsupervised learning of specifications [97, 116]; probabilistic matching [30, 151]; and heuristic-based, which is the similarity-based matching with statistical estimation of property mappings [4, 112]. The first two families have a limitation in scalability because they either repeatedly browse the data or memorize all computations. Meanwhile, the third one is more scalable due to its simple architecture.

We develop *ASL* [108] (automatic schema-independent interlinking), a system classified into the third family, which is the automatic approach. *ASL* can work with any repository with any schema. Therefore, it is a schema-independent system. *ASL* is built upon the architecture of *ScSLINT*. The notability of *ASL* includes two aspects. The first one is the detection of property mappings using a heuristic of overlapping measure on the values of instances. The second one is the property-driven token-based blocking for discarding dissimilar candidates. The former is a solution for the heterogeneity of schemas and the latter is installed for the scalability.

4.2 Problem statement

Given two repositories: the source R_S and the target R_T. Each repository consists of a *multiset* of instances and a schema - the set of properties. The goal of *ASL* is to find

the sub-multiset I of the Descartes product $R_S \times R_T$ so that every member $\langle x, y \rangle$ of I is a pair of coreferent instances and none of the complement set $R_S \times R_T \setminus I$ is coreferent.

Schema-independent instance matching obtains the achieves goal on the repositories whose schema description is not given in advance. The schema of a repository is defined as the set of all properties used to describe the instances. Therefore, in schema-independent instance matching, the property mappings need to be detected using an automatic method.

4.3 *ASL* system

ASL is firmly built upon the architecture of *ScSLINT*. The number of components as well as the function of each component is similar to that of the respective base one. The difference is that *ASL* automates the components by its manners. In the next sections, we describe the detail of each component.

4.3.1 Property alignment generator

The goal of this step is to detect the alignments between the properties in the source and the target schema, both of which are respective to the input repositories R_S and R_T. These alignments are expected to have descriptions about the same attributes of instances. This step contains two smaller sub-steps: property selection and alignment.

4.3.1.1 Selecting properties in source repository

This first sub-step applies a filter on the source schema. This filter removes the unnecessary properties to reduce the noisy information and enhance the speed of later components. A property is important if it covers at least an acceptable portion of the data and has diversity among its values. Therefore, we utilize two metrics to determine the importance level of a property: coverage (*cov*) and discriminability (*dis*), derived from [148]. Equations 4.1 and 4.2 define these metrics in detail.

$$cov(p_k) = \frac{|\{x | x \in R_S, p_k(x) \neq \emptyset|}{|R_S|} \tag{4.1}$$

$$dis(p_k) = \frac{|\cup_{x \in R_S} p(x)|}{|\{x | p_k(x) \neq \emptyset\}|} \tag{4.2}$$

Table 4.1: 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 with a uniform format
URI	original value

In above equations, p_k is the property of interest. x is an instance and $p_k(x)$ is the value of x on the property p_k. To select the important properties, thresholds α and β are used for the coverage and discriminability, respectively.

It is not difficult to determine the appropriate values for α and β. We can expect to have a high value for α and β to obtain the properties that cover a large portion of the repository and well discriminate the ambiguous instances. However, since *ASL* compares multiple attributes of two instances for further steps, α and β can be relaxed to increase the opportunity of the useful properties. The number of important properties in the result can be a criterion to select α and β. In our experiment, we also configure α and β by using this value.

4.3.1.2 Aligning the selected properties into the target schema

This second sub-step finds the corresponding properties in the target schema for each important property of the source schema. The format of an alignment is $\langle p_S, p_T \rangle$, where p_S and p_T belong to the source and the target schema, respectively. The usefulness of an alignment is determined in accordance with its confidence. The confidence $conf$ of $\langle p_S, p_T \rangle$ is estimated by using the coverage of the values of p_T against p_S, as defined in Equation 3. The alignments whose confidence is higher than threshold γ are selected to construct the output of this step.

$$conf(\langle p_S, p_T \rangle) = \frac{|V_{p_S} \cap V_{p_T}|}{|V_{p_S}|} \tag{4.3}$$

$$V_{p_k} = \bigcup_{x \in R_k} \bigcup_{v \in p_k(x)} E(v)$$

In the above equation, E is a value extraction function. For each value, E returns its pre-processed values, in accordance with the type of the corresponding property. The type of a property is assigned by the most major type of its values. *ASL* categorizes values into string, decimal, integer, date, and URI. For a string, E extracts the tokens with stop-word removal. For a decimal, E returns the rounded value with precision 0.01. For a value of the remaining types, E keeps the original value. The summary of E is given in Table 4.1

All values described by p_S and p_T are not compared directly. Instead, their values are firstly preprocessed by function E and cumulated into O_{p_S} and O_{p_T}. After that, O_{p_S} and O_{p_T} are used for comparison. Since the differences between the representations of facts in different repositories are frequently exist, a preprocessing function like E is helpful to detect the expected alignments but having such issue. For example, geographic locations (decimal type) described in coreferent instances are slightly different. E will round these values with 0.01 decimal precision to increase the opportunity of a positive comparison result.

Similar to α and β, γ can be selected by using the number of accepted property mappings. The automatic selection of α, β, and γ requires applications of heuristic or supervised learning and is considered as a future work. In experiment, we evaluate the sensitivity of the system to the parameters by setting the same parameters for many datasets.

The heuristic used in this step includes the selection of useful properties and the alignment. A property is considered as useful if it is used to describe the common attribute of instances and the the described values are diverse. Such properties usually are or nearly the *key* of the given repository. That is, it simultaneously can help discriminate the instances and can cover a large portion of the repository. The alignment of properties is based on a simple overlap measure with the assumption that the properties sharing many similar values are describing the same attribute.

Some systems also applied similar techniques to find the property mappings [4, 112]. However, the prior property selection from source repository was not investigated although the computation of the confidence for all pair-wise mappings between properties is impractical on large dataset. In addition, other systems use Jaccard index to measure the confidence, by replacing the denominator of Equation 4.3 by $|V_{p_S} \cup V_{p_T}|$. Quantifying such union is expensive because all elements of V_{p_T} must be retrieved. As the repository becoming large, especially when the target is usually considered as the referent repository, querying V_{p_T} is not trivial. We only use V_{p_S} for the denominator to improve the scalability.

The goal of this step is to find the property mappings that are useful for matching instances, rather than finding the exact mappings like schema mapping task. For example, considers the 'person' domain, 'label' can be aligned with not only 'label', but also 'full name', 'first name', and 'last name' because these alignments are useful.

The results of this step are later used by ASL to finish the matching task without any other information from the schemas. Since ASL detects the property mappings using the observation on values and does not use the description of schemas, the system can work with repositories whose schema is unknown and therefore ASL is schema-independent.

4.3.2 Blocking

Blocking step generates the pairs of instances that are potentially coreferent. Such pairs are called candidates. This step creates a candidate from two instances if they share at least one value among the first K tokens of the two strings described by the pair of two properties residing in one of previously detected property mappings. Because the order of tokens are not considered, larger value of K can increase the number of candidates. This blocking strategy can be called attribute-driven token-based blocking, where 'attribute-driven' implies the comparison on instances using the property mappings. Theoretically, any type of property can be used for blocking by comparing the returned value of E. However, ASL only uses strings, and especially, instead of taking all tokens, only the first K tokens of strings are needed. The experimental results strongly support the generality of this blocking method.

The objective of this step is to speed up the whole matching process. Although the alignments detected in the previous step have already enabled the verification for any pair of instances, it is extremely expensive to check every combination, especially in large repositories. Thus, blocking is designed as an important step, though its role is simply as an intermediator.

Token-based blocking is one of the most effective techniques so far. The most similar technique to token-based blocking is prefix blocking, which is used in Zhishi.Links [117] and ADL [58]. This blocking method accepts a pair of instances if they share at least an expected number of first tokens in the same order. The token-based blocking of ASL does not consider the order of the tokens because token reordering may occur. In addition to token-based blocking [120], many other approaches, such as canopy clustering [88], sorted neighborhood [50, 166] can be used. However, token-based blocking is much more efficient than the others because it can be easily scaled up by a simple inverted index structure, which $ScSLINT$ supports.

When labeled data is given, some supervised learning method can be used to learn a blocking function that can optimally reduce the number of unnecessary candidates. Two representatives of this approach are adaptive blocking [166] and optimal hashing scheme [29]. Also focusing on eliminating unnecessary candidates but to skip the requirement of labeled data, some systems use token-based ranking [116, 155], which includes a weighting scheme equipped with TF-IDF or BM25. Token-based ranking is more efficient than pure blocking because it assigns a score to the candidates, and therefore supports the mechanism to select the top candidates. However, token-based ranking ignores many correct candidates, especially when the data are highly ambiguous. Therefore, ASL makes the trade-off for better accuracy by using the described blocking method.

4.3.3 Similarity function generator

ASL uses existing similarity metrics supported by *ScSLINT*. For each property mapping, depending on the data type, suitable similarity metrics are assigned to produce the similarity functions. Concretely:

- For string type, Levenshtein and TF-IDF cosine are used. Levenshtein is applied only for short string. *ASL* considers a property as short string if the average tokens of the values described by this property is less than 10. Otherwise, a property is considered as long string.
- For numeric type, *ASL* uses reverse difference.
- For URI and datetime, exact matching is used.

The similarity functions are input into the next step to create a specification used by later components.

4.3.4 Specification creator

In *ASL*, the specification contains S the list of similarity functions F, the similarity aggregator G, and determiner D. The similarity functions are reused from the input of the previous step. The similarity aggregator is defined by the built-in Euclidean distance of *ScSLINT*. The determiner is a filter based on the stable matching principle [43]. The detail of the aggregator and the filter is described in the next two sections.

4.3.5 Similarity aggregator

This step computes the matching score for each candidate. The matching score is the linear combination of the literal similarities between their attributes multiplied with a weight given to the target instance. Equation 4.4 defines the matching score calculation for two instances x_S and x_T.

$$\text{score}(x_S, x_T) = \text{weight}(x_T) \times \sum_{\delta^{\sigma}_{\langle p_S, p_T \rangle} \in F} \max_{u \in p_s(x_S), v \in p_T(x_T)} \sigma^2(u, v) \tag{4.4}$$

where

$$\text{weight}(x_T) = \log_{|arg\,max_{x \in R_T}|x||)}(|x_T| + 1) \tag{4.5}$$

In this equation, δ is the similarity function created from the property mapping $\langle p_S, p_T \rangle$ and the similarity metric σ. In the case that p_S or p_T describes multiple values, the similarity of the most similar objects is used. This mechanism is reflected by the max operator in Equation 4.4. The weighting of the target instance is to prioritize the instances containing more properties. In Equation 4.5, $|x|$ is the number of properties used to describe x. In large repository, this weight is helpful to deal with the situation of self-coreference, when coreferences exist within a repository. In this case, the instance that contains mo more information should be linked first because the further filtering mechanism of the determiner may not retain all those instances.

ASL uses a linear quadratic for similarity aggregation. Compared to averaging, which is widely used by other systems such as SLINT+ [112] and ADL [58], quadratic aggregation has advantage in matching instances when a few attributes of them are missing. For example, when two instances are highly similar on 'latitude' and 'name' but one of them does not contain information of 'longitude', these instances still have high possibility to be coreferent. Because quadratic aggregation aggregates the final score from the square power of 'latitude' and 'name', it returns higher score than averaging. The second case is when incorrect property mappings are used to compute the similarities. Frequently, in this case, the similarities on incorrect alignments are smaller than those of correct alignments. Therefore, with quadratic aggregation, the similarities on correct alignments can contribute more to the final matching score. Meanwhile, conjunctive logic clauses, as used in L2R [135] is the strictest strategy because it only accepts pairs whose each similarity is higher than an associated threshold. For above reasons, we install the quadratic aggregation in *ASL*.

4.3.6 Determiner

This step produces the final coreferences. In *ASL*, the determiner acts as a stable filter. Using this filter, the possibility to be coreferent for each candidate is not directly concluded from its matching score. Instead of that, the matching scores of all related candidates are considered. This filter is based on the principle of stable marriage problem [43], which was evaluated to be effective on instance matching [99, 112]. A candidate (x, y), where $x \in R_S$ and $y \in R_T$, is eventually coreferent if the matching score $\text{score}(x, y)$ satisfies the following conditional statement:

$$\text{score}(x, y) \geq \max(\max_{z \in R_S} \text{score}(z, y), \max_{t \in R_T} \text{score}(x, t)) \tag{4.6}$$

Together with applying the filter, ASL selects N candidates having the highest scores for the result. As the default, N is set to R_S for the assumption that the target is a complete reference.

The basic idea of the filter 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 exists other pairs (x, z) with higher similarity.

4.4 Experiment

4.4.1 Experimental design

We conducted a total of three experiments. Two datasets are used for these experiments, OAEI2011 and DF246. OAEI2011 is the dataset used in OAEI 2011 instance matching track. DF246 is a newly created dataset that consists of 246 subsets sorted increasingly in size. The repositories of this dataset are related to 246 domains of DBpedia and 35 domains of Freebase. We will describe the detail of these datasets later.

In the first experiment, we evaluate the blocking component and analyze the change of the result when varying the numbers of the first tokens K. We use the 193 smallest subsets of DF246 to limit the size of input data at 10×10^9, since the blocking step generates a huge number of candidates in large repositories. We also compare the runtime of blocking step with various value of K.

The second experiment evaluates the final result of ASL. We applied a top-down selection based on matching scores. Concretely, top N pairs with highest matching score are selected. For each subset, the number of output pairs N is equalized to the number of expected pairs. In the case that the score of the N^{th} candidate is equal to $(N + k)^{th}$, ASL outputs k additional pairs as well. We also compare the result of ASL with other recent systems, including two automatic systems, PARIS [151] and Knofuss [116], and three manual systems, SERIMI [3], AgreementMaker [27], and Zhishi.Links [117]. We execute PARIS and Knofuss on DF246 and use the reported results of SERIMI, AgreementMaker, and Zhishi.Links on OAEI2011 [37]. Note that SERIMI was designed later as an automatic system, but at OAEI 2011, the system used prior knowledge about the schema [3, 5]. PARIS provides a probabilistic-based matching which is considered as an innovative approach among automatic systems. Knofuss is an unsupervised learning-based system, which is reported to enhance the result on some datasets. We are interested in seeing how it performs on DF246 and OAEI2011. Zhishi.Links obtained

the best result on OAEI2011 dataset and is still considered as the best performed system. We also include the result of SERIMI and AgreementMaker on this dataset as a reference.

The following are some notes for running PARIS and Knofuss. For PARIS, we use the default parameter for the number of iterations. The same top-down selection strategy as is used for *ASL* is applied and only the instance mappings are considered, instead of all types of generated ones (e.g., properties and literals). For Knofuss, we extend the default value of the maximum Lucene search results from 20 to 500 because we expected to have high pair completeness. We also increase the maximum number of genetic algorithm generations from 20 to 100 for a more reliable result. Since Knofuss is comparably slow, we limit the execution time to 3 days for each subset. In addition, because the matching score is not available for the output of Knofuss, we select all the generated pairs for computing the pair completeness, precision, and F1.

Finally, in the last experiment, we analyze the scalability of the tested systems. We report the detailed execution time and memory footprint of *ASL*, PARIS, and Knofuss.

4.4.2 Evaluation metrics

We use the pair completeness, precision, and their harmonic mean $F1$ to evaluate the instance matching result. For the blocking, as the objective of this step is to find the set of candidates that is small but contains as many correct ones as possible, the pair completeness and the reduction ratio are used as the measures for quality and quantity of generated candidates, respectively. In Section 2.5, we describe the detail of these metrics.

For evaluating the final result of instance matching, a high pair completeness reflects the ability of detecting as many expected correct pairs as possible. Meanwhile, high precision expresses that detected pairs contains a small number of incorrect ones. Similarly, high pair completeness for blocking step guarantees that correct candidates are retained as many as possible. Also, reduction is very important because it reflects how many unnecessary candidates are discarded.

4.4.3 Datasets

DF246 and OAEI2011 datasets contain portions of DBpedia, Freebase, NYTimes, and Geonames. Because *ASL* is a schema-independent system, DF246 is chosen due to its variety of schemas. In addition, OAEI2011 is used to compared *ASL* and other non-learning based systems. We set up a few adjustments for the data of these data sources

because they contain some information that is not related to the instance matching problem or that reduces the data quality. For DBpedia, we use only the triples that are loaded into DBpedia's SPARQL endpoint[1] and describe 'resource' instances[2]. For Freebase, we only consider the 'topic' instances[3] and the triple with 'm' identifier[4]. In addition, for DBpedia and Freebase, we integrate the redirect instances into their reference. Next, we describe the datasets in detail.

4.4.3.1 DF246 dataset

DF246 is created for evaluating the generality of *ASL*, including the accuracy and the scalability. This dataset contains 246 subsets combined from 246 source repositories and 35 target repositories. The creation process for this dataset consists of the two following steps.

Step 1: Select source repositories from DBpedia. To have many repositories with different schemas, we divide DBpedia into smaller parts based on existing domain information of the instances. We split the repositories with assumption that R_S is a single domain repository. Since no clear definition for 'domain' exists due to the natural hierarchical relations of concepts, our assumption is close to that R_S does not contain so different domains in terms of schema. For example, the schemata of university and college (educational institution domain) are highly equivalent, whereas those of ship and train (transportation domain) are very different. In fact, this assumption is not strict. Dividing the source repository into separable domains is a feasible inexpensive task. In DBpedia, an instance may belong to different domains that are involved in a "parent-child" relation. Therefore, a domain, which consists of a set of instances, may have one parent and many children.

Since we split the repositories with above assumption, we merge domain C_1 into C_2 if the schemas of them are similar. In order to achieve that objective, we conducted a *schema conformance* check for each sub-domain relation[5]. Domain C_1 is conformable to its parent C_2 if every property p satisfies the condition in Equation 4.7.

$$|f(p, C_1) - f(p, C_2)| \leq 0.5 \tag{4.7}$$
$$f(p, C_k) = \frac{|\{x|\langle s, p, o\rangle \in x, x \in C_k\}|}{|C_k|}$$

[1] http://wiki.dbpedia.org/DatasetsLoaded
[2] The triples whose RDF subjects start with http://dbpedia.-org/resource/
[3] The instances that contains a triple whose value is http://rdf.freebase.com/ns/common.topic
[4] The triples whose RDF subjects start with http://freebase.-com/ns/m
[5] http://mappings.dbpedia.org/server/ontology/classes/

Table 4.2: Overview of DF246 dataset

	# Sources	# Targets	# Expected
Min	127	1,804	126
Max	602,293	25,625,291	597,566
Avg.	10,754.1	2,002,283.6	9,339.6

Using the checking result together with the original sub-domain specification, we derive the priority for each domain to classify the instances. That is, a domain is initially given higher priority than its parent in order to diversify the dataset. However, if a domain is totally conformable to its parent, the higher priority is given to the parent instead.

In DBpedia, we extract the domain information of instances by using the property rdf:type. Among 398 domains with respective DBpedia 3.9 (English) instances, 46 classes are conformable (e.g., governor and politician, bone and anatomical structure). Note that identical domains are counted as one (e.g., http://dbpedia.org/ontology/Place and http://schema.org/Place). Also, not every instance is classifiable because the type information of 584,520 instances is missing. Finally, 352 domains are unconformable and resulted in 352 repositories.

Step 2: Select target repositories from Freebase. We first separate Freebase (2013/09/03) into different repositories by using the domain information of the instances, given by the property fb:type[6]. After that, we map each source repository into one target that shares the most coreferent instances, as declared in the gold standard[7]. Finally, we remove the mappings having less than 100 coreferent pairs.

The first task extracted 35 different repositories, the second task obtained 2,668,372 standard pairs, and the last task produced 246 subsets. The 246 repositories constructed from DBpedia, 35 repositories constructed from Freebase, and the mappings between them are listed in Appendix A, B, and C, respectively.

We sort these subsets increasingly by their size. The size of a subset is the product of the number of instances in the source and the target repositories. Table 4.2 is the summary of the number of instances in these subsets. In this table, #Sources and #Targets represent the number of instances in source and target repositories. #Expected is the number of actual coreferent pairs. Note that each Freebase instance refers to a topic and can belong to multiple sibling domains (e.g., book, music, and location). Therefore, overlaps are found between almost every two of the 35 extracted repositories.

[6] http://rdf.freebase.com/ns/type.object.type
[7] http://downloads.dbpedia.org/3.9/links/freebase_links.nt.bz2

Table 4.3: Overview of OAEI2011 dataset

ID	Target	Domain	# Sources	# Expected	# Originals
D1	DBpedia	location	3,840	1,917	1,920
D2	DBpedia	organization	6,088	1,922	1,949
D3	DBpedia	people	9,958	4,964	4,977
D4	Freebase	location	3,840	1,920	1,920
D5	Freebase	organization	6,088	3,001	3,044
D6	Freebase	people	9,958	4,979	4,979
D7	Geonames	location	3,840	1,729	1,789

4.4.3.2 OAEI2011 dataset

OAEI2011 is used to compare *ASL* with other manual systems. The gold standard of this dataset was presented at the OAEI 2011 instance matching track [37]. The source repositories are three separated domains of NYTimes: location, organization, and people. The targets are full DBpedia, Freebase, and Geonames. To construct the repositories, we use DBpedia 3.7 (English), Freebase 2013/09/03, and Geonames 2014/02. The numbers of instances are 4,183,361 for DBpedia, 40,358,162 for Freebase, and 8,514,201 for Geonames.

An important note for this dataset is that we detected some incorrect and missing pairs in the previous standard as compared to the current actual data. In evaluation, we do not count the incorrect, the missing, and erroneous pairs. Consequently, the numbers of expected pairs are slightly different from the original pairs published in 2011. We summarize the size of the OAEI2011 dataset in Table 4.3.

Compared to the more recent OAEI datasets, the OAEI2011 dataset is more suitable for our experiment because of the similar benchmark objective. In year 2012, OAEI provided the training and test split for OAEI2011 dataset in order to evaluate learning-based systems. The OAEI dataset of year 2013 and 2014 are small and designed for special challenges (e.g., multiple languages, string distortion, value deletion, and non-uniform representations). Since *ASL* focuses on real and large data, OAEI2011 is suitable for the experiment.

All datasets, including the repositories, the labels for expected pairs, and their detailed descriptions as well as the *ASL* source and the full experimental results, are available at http://ri-www.nii.ac.jp/ASL/.

4.4.4 Experimental environments

All the executions for *ASL* are conducted on a Windows machine equipped with one Intel Core i7 4770K CPU and 16 GB memory. The environment for running PARIS

and Knofuss is a FreeBSD machine with two Intel Xeon E5-2690 CPUs and 256 GB memory. We run PARIS and Knofuss on a high-performance computer because the current versions of these systems require a large amount of memory.

4.4.5 Parameter settings

ASL has three parameters: α, β, and γ. These values are used in the first step, property mapping. For checking the sensitivity of the system to the parameters, we use the same parameters for all subsets. We permanently set $\alpha = 0.3$, $\beta = 0.75$, and $\gamma = 0.7$. We assign the value 0.75 to β in the expectation that at least 75% of the values are unique. By fixing β, the value of α is selected as the maximal value that accepts at least one property describing an arbitrary string attribute for every subset of DF246. Similarly, γ is set to the maximal value that retains at least one alignment between string attributes for every subset.

Using the above parameters, we obtain the results as shown in Table 4.4. In this table, P_S and P_T are the number of all properties in source and target repository, respectively. M is the number of property mappings while P_S^* and P_T^* are the properties aligned to construct these alignments. Because one property may be aligned with many others, M, P_S^*, and P_T^* thus can be different. For DF246, we report the values of minimum, maximum, and average because the number of subsets is high. For OAEI2011 we report the detailed results of each subset According to this table, although the source and target schemas for most subsets have many properties, not many of them are usable for the instance matching task. On average, only 4.23% properties of source repositories in DF246 are included in the property mappings, when using the configured parameters. It reflects that many properties are not useful for repositories in DF246 dataset. In other words, in this dataset, the properties are frequently used to declare the information that is not shared by source and target repositories. The mapped properties in the target schema are even lower, at only 0.11%.

For every subset, at least the 'label' properties are aligned on both DF246 and OAEI2011. For DF246, the 'description' properties are also frequently mapped, while other properties are different between subsets, depending on the domain. For OAEI2011, the latitude and longitude are also correctly mapped in the location domains (D1, D4, and D7) as well as the alignment between the labels. The 'label' property of NYTimes is aligned more than one time because the target repositories have multiple properties that describe the same information.

For evaluating the quality of property mappings, experts are asked to check the property mappings. For OAEI2011, all generated property mappings are correct. For DF246, we

Table 4.4: Property alignment results

		P_S	P_T	P_S^*	P_T^*	M
DF246	Min	32	753	1	1	1
	Max	4,144	2,439,785	17	9	45
	Avg.	435.74	64,300.24	7.71	5.04	20.85
OAEI2011	D1	22	45,858	3	5	5
	D2	20	45,858	1	3	3
	D3	20	45,858	1	5	5
	D4	22	2,455,627	3	5	5
	D5	20	2,455,627	1	3	3
	D6	20	2,455,627	1	4	4
	D7	22	14	3	4	4

Table 4.5: Blocking results with respect to K first tokens

	$K=1$		$K=2$		$K=3$		$K=4$		$K=\infty$	
	pr	rr	pc	rr	pc	rr	pc	rr	pc	rr
Min	0.9866	0.8602	0.9923	0.8423	0.9923	0.6938	0.9923	0.5435	0.9953	0.1226
Max	1.0000	0.9999	1.0000	0.9991	1.0000	0.9998	1.0000	0.9972	1.0000	0.9263
Avg.	0.9995	0.9965	0.9998	0.9880	0.9999	0.9696	0.9999	0.9533	0.9999	0.6064
St.dev.	0.0015	0.0127	0.0009	0.0213	0.0006	0.0510	0.0006	0.0723	0.0004	0.1646

randomly select 35 subsets related to 35 target repositories of DF246 dataset. In total, 48 (10%) out of 480 property mappings are incorrect and occur on 9 subsets. For example, on the subset of 'hollywood cartoon' (DBpedia) and 'film' (Freebase), 'release date' and 'birth date' is aligned. This case happens because the 'film' repository shares many instances from the 'people' repository, so that the overlap measurement (Equation 3) may produce such an alignment due to the diversity of the target. Although there are datasets for schema mapping problem, such datasets are not applicable for evaluating property mapping step. Schema mapping finds exactly correct alignments while *ASL* focuses on alignments that can be used to match instances. Therefore, we manually evaluate this step instead of using schema mapping datasets. In the second experiment, we compare the result of instance matching using all generated alignments and using only correct alignments, on 35 selected subsets.

4.5 Results and discussions

4.5.1 Experiment 1: Blocking

Table 4.5 shows the result of the blocking step with a varying number of tokens.

K=∞ implies that all tokens are used. According to this table, as the tokens order is not considered, the more tokens are used, the more candidates are generated. K=1 is

Table 4.6: Running time of blocking step with different K values (unit: sec)

	$K=1$	$K=2$	$K=3$	$K=4$	$K=\infty$
Min	0.01	0.06	0.12	0.11	0.69
Max	64.93	148.35	1348.55	2474.87	43096.80
Avg.	4.81	13.54	39.93	72.09	2333.95

Table 4.7: Blocking results for K=1

		pc	#Candidates	*rr*
DF246	Min	0.9866	2,600	0.8602
	Max	1.0000	5,149,781,137	0.9999
	Avg.	0.9995	87,372,412	0.9965
OAEI2011	D1	0.972	2,054,099	0.9999
	D2	0.988	4,592,818	0.9998
	D3	0.997	20,563,742	0.9995
	D4	0.988	9,386,067	0.9999
	D5	0.977	19,717,567	0.9999
	D6	0.991	67,232,889	0.9998
	D7	0.967	3,785,063	0.9999

sufficient due to its competitively high pair completeness (*pc*), as well as best and stable reduction ratio (*rr*). Compared to K=∞, only 213 (0.054%) more correct candidates are missed in total, whereas the reduction ratio greatly increases at 32.44% for the total of 193 subsets. The numbers for K=2 are 130 and 0.81%, respectively. Although 0.81% is not a high value when considering the quantity, the number of candidates is 3.29 times different with more than 2×10^9 unnecessary candidates included.

Table 4.6 shows the runtime of blocking step when changing K. As increasing K, the required time increases quickly and reach long responding time on large subsets when $K \geq 3$. Furthermore, the runtime of verification step depends on the number of candidates and it is very expensive in overall if blocking step generates too many candidates. In summary, K higher than 1 costs much time and includes many useless candidates while not compensating much on pair completeness. Therefore, we use K=1 for the remaining experiments.

The detailed result for using K=1 is reported in Table 4.7. The result shows that only the first token is sufficient for blocking not only on the whole DF246, but also on OAEI2011. For DF246, most of the reduction ratios are distributed at a very high value. The average is 0.997 and the standard deviation is narrow at 0.013. The highest number of candidates is more than 5.1×10^9 for the biggest subset 'album' (116,368 instances) and 'music' (25,625,291 instances) repositories. This number is comparably big; however, it is still efficient since 2.77×10^{12} comparisons are ignored.

Table 4.8: Instance matching results

		rec	*prec*	F1
DF246	Min	0.8769	0.8769	0.8769
	Max	1.0000	1.0000	1.0000
	Avg.	0.9750	0.9751	0.9751
	Std.	0.0257	0.0255	0.0256
OAEI2011	D1	0.8779	0.8807	0.8793
	D2	0.8741	0.8709	0.8725
	D3	0.9647	0.9647	0.9647
	D4	0.9021	0.9035	0.9028
	D5	0.9157	0.9169	0.9163
	D6	0.9309	0.9315	0.9312
	D7	0.8785	0.8750	0.8768

Because high pair completeness is obtained in this step, the input for the further steps is almost the 'original full repositories' but has a much more compact size.

4.5.2 Experiment 2: Final result

Table 4.8 is the result of the determiner. This is also the final result of the whole instance matching process. Generally, the result of DF246 is better than that of OAEI2011, due to the rich information given in DBpedia and Freebase. For DF246, *ASL* can totally detect 98.35% (2,259,713 pairs) of the expected pairs and obtained a high F1 for most subsets. Of 170 (69%) subsets obtaining an above-average F1 score (0.975), only 6 subsets obtained an F1 score lower than 0.9. For OAEI2011, since the information given in the source repositories is very limited, it is difficult to discriminate the highly ambiguous instances. For example, 'label' is the only attribute that can be used for D2, D3, D5, and D6. The other subsets are slightly richer since they come with geographic location, although not all instances (93%) contain such information. In addition, the high results indicate the selected parameters work well on tested datasets, which are diverse in domain and schema. This fact supports that the system is not so sensitive to the parameters.

As we discussed, the incorrect property mappings are included on 9 of out the 35 randomly selected subsets. On these 9 subsets, the highest and average F1 are 0.9701 and 0.9945, respectively. We attempt to manually remove incorrect property mappings and input only the correct ones into blocking and candidate verification step. The highest and average F1 after this refinement is 0.9758 and 0.9952, respectively. There is one subset on that the final results are the same between the inclusion and exclusion of incorrect alignments. There are 9 out of 30 property mappings are incorrect on this subset. In summary, it is clear that using only the correct property mappings is recommended for

Table 4.9: Instance matching results of *ASL* and PARIS on 240 subsets of DF246

	Min			Max		
	rec	prec	F1	rec	prec	F1
ASL	0.8769	0.8769	0.8769	1.0000	1.0000	1.0000
PARIS	0.5165	0.5165	0.5165	1.0000	1.0000	1.0000
	Avg.			Std. dev.		
	rec	prec	F1	rec	prec	F1
ASL	0.9750	0.9751	0.9750	0.0259	0.0258	0.0259
PARIS	0.9362	0.9223	0.9290	0.0701	0.0763	0.0722

Table 4.10: Instance matching results of *ASL* and Knofuss on 239 subsets of DF246

	Min			Max		
	rec	prec	F1	rec	prec	F1
ASL	0.8769	0.8769	0.8769	1.0000	1.0000	1.0000
Knofuss	0.0000	0.0000	0.0000	1.0000	1.0000	0.9980
	Avg.			Std. dev.		
	rec	prec	F1	rec	prec	F1
ASL	0.9754	0.9756	0.9755	0.0256	0.0255	0.0255
Knofuss	0.4661	0.6212	0.5185	0.3570	0.3915	0.3643

matching instances. However, in overall, *ASL* still obtains high results on DF246 and OAEI2011, thanks to the dominance of the correct alignments.

A useful property mapping that describes the same Wikipedia page of two coreferent instances is detected on 169 subsets (69%) of DF246. Concretely, the average confidence (Equation 3) of this alignment is 0.86. That is, probabilistically, only 86% of expected coreferent instances share the same value linking to Wikipedia page. Meanwhile, the lowest and average pair completenesss of these subsets are 0.8772 and 0.9844, respectively. On the remaining 77 subsets (31%), the lowest and average pair completenesss are 0.8769 and 0.9544, respectively. These results suggest that using property mapping on Wikipedia page can improve the result on DF246. However, this alignment is not the main decisive factor for obtaining the high results because the average pair completeness on above 169 subsets is higher than 0.86 and that on remaining 77 subsets is also at high value.

For further evaluating *ASL*, we compare the results of *ASL* and other systems. The details are described as follows.

4.5.2.1 Comparison to automatic systems

Table 4.9 and Table 4.10 compare the results on DF246 of *ASL* and PARIS, and *ASL* and Knofuss, respectively.

Our machine cannot execute PARIS on 6 subsets of the 'book' target repository, because the number of properties is huge, 2.4×10^6. Meanwhile, Knofuss does not respond after 3 days of running on 4 subsets, whose source and target repositories contain more than 10^5 and 10^6 instances, respectively. Knofuss also generates an empty result on 3 subsets. We separate the comparisons between *ASL* with PARIS and Knofuss. For each comparison, we respectively collect the result on the subsets that are applicable for PARIS and Knofuss.

Both *ASL* and PARIS achieve good results with a high F1 and a narrow standard deviation on average. *ASL* is better for most cases as it outperforms PARIS on 192 (80%) subsets. For all subsets together, *ASL* obtains 0.9837 for F1, whereas PARIS obtains 0.9227. For each subset, the paired t-test yields a significant difference ($t = 10.2317$, $p < 0.0001$).

Since Knofuss does not output the matching scores, we consider all detected pairs of Knofuss as the final results. The result of Knofuss is generally lower than that of *ASL* and PARIS. On 239 applicable subsets for Knofuss, the F1 scores of *ASL* are significantly better than those of Knofuss on 232 subsets ($t = 19.7260$, $p < 0.0001$) and only on 7 subsets Knofuss outperforms *ASL*. The F1 scores of Knofuss are under their average on 113 subsets and no correct pairs are detected on 43 subsets. For those 43 subsets, Knofuss offers a heavy weight for the instance consisting of many triples. Therefore, many instances of the source are linked to only one big instance of the target but none of them are correct. The low pair completeness also results from the fact that Knofuss frequently generates numbers of pairs that are fewer than expected. Only 63.6% of the total 1,253,254 expected pairs are generated. However, the precision is modest, at 62.12% on average.

4.5.2.2 Comparison to manual systems

Table 4.11 compares the F1 scores of *ASL* on OAEI2011 with SERIMI [3], AgreementMaker [27], and Zhishi.Links [117]. According to this table, *ASL* performs better on most subsets than do SERIMI and AgreementMaker. Compared to Zhishi.Links, *ASL* is a competitive runner-up. Note that Zhishi.Links includes some effective customizations for OAEI2011. The system uses 19 unification rules for strings (e.g., 'Co' and 'Company', 'Bronx', 'Manhattan', and 'NYC'). Meanwhile, *ASL* and other systems use a general similarity measure without any specific adaptation.

The reported result of other systems on this datasets is based on a much smaller dataset simplified from the original data due to the scalability issue. Using the simple version of this dataset is not difficult to obtain near perfect results, as reported in [112]. Although

Table 4.11: F1 scores of *ASL* and other manual systems on OAEI2011

	ASL	SERIMI	AgreementMaker	Zhishi.links
D1	*0.88*	0.68	0.69	**0.92**
D2	0.87	*0.88*	0.74	**0.91**
D3	*0.96*	0.94	0.88	**0.97**
D4	*0.90*	**0.91**	0.85	0.88
D5	**0.92**	*0.91*	0.80	0.87
D6	*0.93*	0.92	**0.96**	*0.93*
D7	*0.88*	0.80	*0.85*	**0.91**

Table 4.12: Detailed runtimes of *ASL*
Unit: %, except Total column.

		Property mapping	Blocking	Similarity aggregator	Total (unit: sec)
DF246	Min	0.01	0.13	26.21	2.33
	Max	31.92	19.96	99.16	83314.2
	Avg.	3.16	1.31	91.41	2753.5
OAEI2011	D1	19.34	1.65	79.01	295.79
	D2	15.5	0.96	83.53	341.12
	D3	13.49	2.68	83.6	389.12
	D4	14.66	1.58	84.04	924.98
	D5	16.42	3.16	80.7	1022.80
	D6	11.43	2.74	86.07	1356.08
	D7	11.4	7.69	78.72	48.73

we cannot truly compare *ASL* and other systems, the comparison reported here still has it value of a reference comparison. In this specific situation, given more difficult data but *ASL* obtains better result.

The competitive result of *ASL* compared to manual systems not only confirms the good performance of *ASL*, but also supports the position of an automatic approach for instance matching solutions.

4.5.3 Experiment 3

4.5.3.1 Runtime of *ASL*

Table 4.12 summarizes the detailed runtimes of *ASL* on DF246 and OAEI2011. The most expensive step of *ASL* is the candidate verification, which requires an average of 91.4% of the total time, even parallel processing (8 threads) is applied for this step. This fact also reveals the importance of blocking for reducing the number of candidates. Since blocking averagely reduces 99.65% of possible pairs (see Table 5), without this step, it costs approximately 6 years to finish the matching process for DF246, instead

Table 4.13: Total runtimes of *ASL* and PARIS on 240 subsets of DF246 (unit: sec)

	Min	Max	Avg.
ASL	2.33	83,314	2763
PARIS	18	33,019	2298

Table 4.14: Total runtimes of *ASL* and Knofuss on 239 subsets of DF246 (unit: sec)

	Min	Max	Avg.
ASL	2.33	27,684	1,782
Knofuss	50	186,428	12,715

of the current 7.8 days. In addition, it is worthwhile to index the data for speeding up the property mapping and blocking since the time for building the index is very minor compared to the total. The most expensive indexing is over 70% for one small subset, on which *ASL* takes 4 seconds to finish the whole process, including the very fast candidate verification step. The reason is the low number of candidates and only one property to be considered.

4.5.3.2 Comparison to PARIS and Knofuss

We conducted a comparison of the scalability of *ASL*, PARIS, and Knofuss on the applicable subsets of DF246, as we reported in the second experiment. Note that the machine for running *ASL* is not more powerful than that for running PARIS and Knofuss. PARIS is faster than *ASL* in overall (see Table 4.13). *ASL* is built upon *ScSLINT*, therefore it is optimized for the memory footprint. The working set of *ASL* is at most 14 GB on the tested subsets. Meanwhile, the efficiency of PARIS is supported by a working set of 154 GB memory, and more than 250 GB is needed for the 6 subsets that our machine failed to execute. *ASL* is very fast on small subsets, whereas PARIS is more stable and more efficient on large subsets. For example, PARIS is faster than *ASL* on 57 subsets. Of these, 33 subsets are larger than 10^{11}, whereas only 16 of the remaining 183 subsets are as large.

According to Table 4.14, Knofuss is not as fast as *ASL*. The system takes 35 days to finish the benchmark and does not respond after 10 days running on the largest subset. Knofuss also uses more memory, which is 42 GB for the maximum. It is clear that Knofuss needs to be reduced in complexity, whereas PARIS needs memory optimization. *ASL* can also be improved by using more memory to support the concurrent candidate verification steps.

In this section, we reported our experimental results. The results show the high performance of *ASL* in both accuracy and efficiency. *ASL* significantly improves the result on

the DF246 dataset in comparison with the state-of-the-art system PARIS. *ASL* is also competitive with manual systems on the OAEI2011 dataset.

4.6 Summary

Schema-independent instance matching is very important because of its applicability. We develop *ASL*, a system that can work on any repository with any schema. *ASL* automatically detects the property mappings without any description of the schemas. The schema-independent capability of *ASL* is verified by the experiments on more than 250 different pairs of schemas. For those dataset, *ASL* shows its portability, effectiveness, and efficiency. *ASL* also significantly outperforms other automatic systems and is competitively better than manual systems.

Chapter 5

ScLink: Supervised specification-based instance matching

In this chapter, we describe our contribution for supervised specification-based instance matching. We present *ScLink* system, a scalable, effective, and efficient system. We write this chapter with the order of motivation (Section 5.1), problem statement (Section 5.2), the detail of $ScSLINT$ (Section 5.3), and the experiment (Section 5.4).

5.1 Motivation

The early state of instance matching is the manually operated systems [27, 117], in which the specification 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 specification. 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. The matching specification can be effectively and automatically constructed by using learning algorithms. Unsupervised learning is an option for a fully automatic system [12, 47, 101, 116]. 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 specification has the advantages in high accuracy and low complexity. Although this approach requires some labeled instance pairs for the learning step, the small size of training data can compensate the loss of automation. Specification learning has been the focus of a few systems [61, 100, 116]. However, those systems are not suitable for highly heterogeneous and large-scale datasets because they need manual interventions or are at high complexity.

In the previous chapter, we presented *ASL*, a schema-independent instance matching system that works without labeled data. *ASL* achieved good performance but the quality of the result can be far better with supervised methods. In some matching tasks, when the accuracy is at high expectation and training data is available, it is essential to leverage a supervised system. We are motivated from developing a supervised specification-based instance matching system that can handle heterogeneity, scalability, and somehow reduces the issue of ambiguity.

We present *ScLink* [111], 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. Compared to previous supervised systems, *ScLink* contains three main novelties. The first one is *cLearn* [104, 105], a heuristic-based specification 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 coreferent instances. The objective of this step is to avoid taking the comparisons for all pair-wise alignments of instances between input repositories so that it improves the scalability. The optimal blocking model learned by *minBlock* is far better than existing non-learning approach used by other systems. The third novelty is the modified BM25

(*Modified-BM25*), 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 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* [106, 109]. Compared to *cLink*, the scalability, ambiguity, and heterogeneity are more radically solved in *ScLink*.

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 *Modified-BM25*. We re-implement recent specification learning algorithms [58, 60, 61, 100, 116] 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 specification. It supports the applicability of *ScLink* in practical instance matching problems.

5.2 Problem statement

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. The set of attributes of an instance x described by property p is formatted as $p(x)$. Due to the heterogeneity, $p(x)$ may be empty (e.g., missing value), contains one, or multiple elements (e.g., different names of one thing). The objective of instance matching is to identify the coreferent set $I \in R_S \times R_T$. A pair $\langle x, y \rangle$ of instances $x \in R_S$ and $y \in R_T$ belongs to I if x and y co-describe the same object. In supervised scenario, a subset of $R_S \times R_T$ is labeled (positive/negative) and given to the learning algorithms. When the learning process finishes, the obtained knowledge is applied for detecting the coreferences 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 coreferent 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.

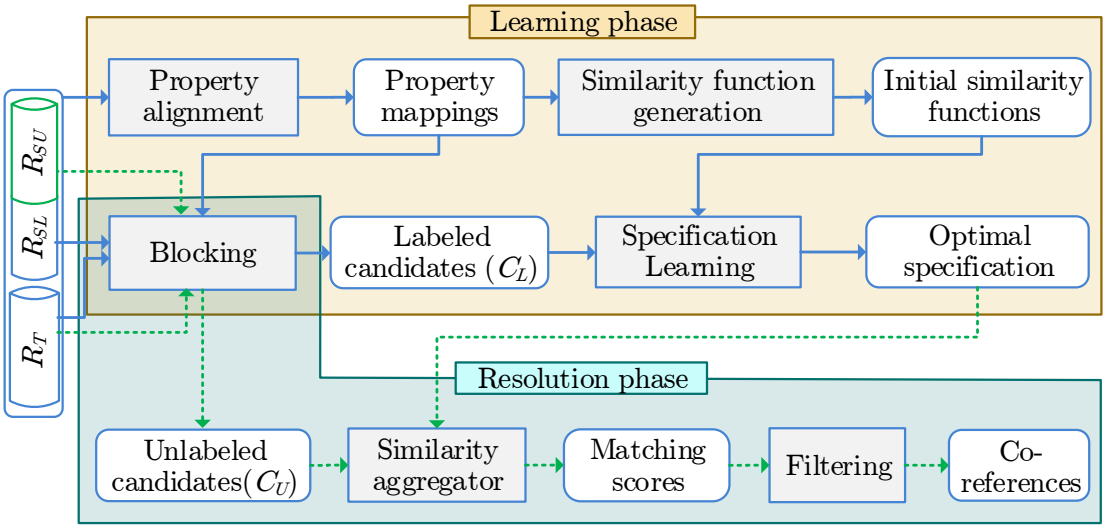


FIGURE 5.1: The general architecture of *ScLink*.

5.3 *ScLink* system

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

5.3.1 Overview

The general workflow of *ScLink* is depicted in Figure 5.1. The instance matching process consists of two phases: learning and resolution. The *learning phase* contains four steps: *property alignment*, *similarity function generation*, *blocking*, and *specification learning*. The *resolution phase* contains four steps: *blocking*, *similarity aggregation*, and *filtering*. *ScLink* is built upon the *ScSLINT* framework (Chapter 3). However, for clearer description of *ScLink*, we merge and split some components of the original *ScSLINT*. The underlying workflow and implementation fit to those of *ScSLINT*. Compared to *ScSLINT*, the *specification creator* is merged to *specification learning*. The *determiner* is named with *filtering* for better describing the specific case of *ScLink*. In addition, the result of *blocking* is separated into labeled and unlabeled, in accordance to the two phases.

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}, where L and U stand for 'labeled' and 'unlabeled', respectively. In *learning phase*, first, the *property alignment* generates the property mappings using statistical analyses over 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 *specification learning*, in which *cLearn* is executed. *cLearn* finds the optimal specification, which contains the specification of further steps, including *similarity aggregation* and *filtering*.

In *resolution phase*, the optimal specification is used to detect the coreferences 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 coreferences for unlabeled candidates by applying a constraint that considers the matching score of all candidates.

Most parts of the above workflow are shared between many specification-based supervised instance matching systems [58, 60, 61, 99]. 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.

5.3.2 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 R_S and then aligns them to the appropriate properties in R_T. This mechanism is similar to that of *ASL*.

5.3.2.1 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 discriminability and coverage are considered. The discriminability of a property p reflects how many unique values are described by p. Meanwhile, the coverage (frq) of p expresses how many instances contain p. These measures are calculated by Equation 4.1 and 4.2, respectively.

We separate the properties by their type before performing the selection. This mechanism enables 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.

5.3.2.2 Align properties between source and target repository

Each selected property from source repository is aligned with the properties having the same type in the target. Each alignment also is called a mapping. The confidence of a mapping $\langle p_S, p_T \rangle$ is measured by counting the sharing values of p_S and p_T, as formulated in Equation 4.3. In that equation, E is a preprocessing function and *ScLink* also uses the function similar to *ASL*. Table 4.1 is the detail of E. 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 addition, 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.

This2 alignment step of *ScLink* is different from that of *ASL*. *ScLink* selects the target properties independently for each property of the source repository. Meanwhile, *ASL* selects the target properties by taking the top most confident alignments over all properties. The reason is, in *ScLink*, there is another learning step to refine the mappings and *ASL* does not have that step. Therefore, *ASL* uses the heuristic in a limited space for not including many unwanted mappings. In contrast, *ScLink* tends to include more useful mappings even it can accept the incorrect mappings as well.

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 *specification learning* step.

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.

5.3.3 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

TABLE 5.1: 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

of similarity functions. The execution of them is conducted by the *similarity aggregation* step, which is described by Section 5.3.6. We denote a similarity function as $\delta^{metric}_{mapping}$. For example, $\delta^{Exact}_{[name,label]}(x,y)$ and $\delta^{Jaro}_{[name,label]}(x,y)$ compare the pair-wise of $name(x)$ and $label(y)$ using exact matching and Jaro similarity, respectively. The formal definition of a similarity function is given in Equation 5.1.

$$\delta^{\sigma}_{\langle p_S,p_T\rangle}(x,y) = \max_{v_x\in p_S(x),v_y\in p_T(y)} \sigma(v_x,v_y) \tag{5.1}$$

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 (e.g., 'Sony' and 'Sony Corporation' are names of a company and both can be described in the same instance).

ScLink assigns only the similarity metrics that are suitable for the type of the interested mapping. The list of similarity metrics is reported in Table 5.1. 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 [24], the focus of *ScLink* is the general context. For *number*, reserve difference is used to calculate how much close the two input numbers are. We apply Levenshtein metric for short strings comparison because it is effective for this kind of data [79]. For long string, 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 (TF) and the inverse document frequency (IDF). Equation 5.2 and 5.3 are the customized calculations of those weights, for adapting with the instance matching scenario.

$$TF(w,x,R) = \frac{\sum_{p\in schema(R)} |\mathcal{V}(w,p,x)|}{\max_{t\in R} \sum_{p\in schema(R)} |\mathcal{V}(w,p,t)|} \tag{5.2}$$

$$IDF(w,R) = \log \frac{|R| - \sum\limits_{p \in schema(R)} |\{x | x \in R, \mathcal{V}(w,p,x) \neq \emptyset\}|}{\sum\limits_{p \in schema(R)} |\{x | x \in R, \mathcal{V}(w,p,x) \neq \emptyset\}|} \tag{5.3}$$

where

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

By including 5.2 and 5.3, 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 Equation 5.2 and 5.3, respectively.

$$cosine(a,b) = \frac{\sum\limits_{w \in a \cap b} TFIDF(w,x,R_S) \times TFIDF(w,y,R_T)}{\sqrt{\sum\limits_{w \in a} TFIDF(w,x,R_S) \times \sum\limits_{w \in b} TFIDF(w,y,R_T)}} \tag{5.4}$$

$$TFIDF(w,x,R) = TF(w,x,R) \times IDF(w,R)$$

$$mBM25(a,b) = \frac{|a \cap b|}{|a \cup b|} \times \sum_{w \in a \cap b} tWeight(w,x,y) \times tEdit(w,a,b) \tag{5.5}$$

where

$$tWeight(w,x,y) = IDF(w,R_S) \times IDF(w,R_T) \times \frac{TF(w,y,R_T) \times k_1}{TF(w,y,R_T) + k_2}$$

$$tEdit(w,a,b) = invDiff(pos(w,a) - pos(w_0,a), pos(w,b) - pos(w_0,b))$$

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 [132], 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 experiment, the modified BM25 shows its considerable effectiveness against the TF-IDF Cosine on highly ambiguous datasets.

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.

5.3.4 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 coreference. 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 coreferences, 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. According to the experiment of *ASL* in Section 4.5.1, token-based blocking using only the first element is sufficient for real datasets. Therefore, in *ScLink*, the blocking mechanism also 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 $B_{def}(R_{SL} \times R_T)$. 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 [30, 154]. 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 coreferences. For the learning of such model, we propose *minBlock*. *minBlock*

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, F(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
```

considers all input blocking functions as baseline and tries to remain the actual coreferences 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 Equation 2.24 (Section 2.5). The idea of *minBlock* is simple. In each main iteration controlled by line 3, it removes one blocking function that generates the most unexpected candidates but does not increase the recall from an acceptable value. This 'acceptable' recall is defined by comparing with the baseline model and is managed by t_{loss}. By default, t_{loss} is set to 1.0 for guaranteeing zero loss in recall.

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 $O(\frac{n(n+1)}{2})$, where n is the size of B_{def}. In average, the complexity is not accurately estimated because it depends on the unknown probability of the **if** statement. However, the complexity in the worst case indicates a moderate computational effort so that it is reasonable to use *minBlock* for supervised instance matching.

The learning of blocking model is also investigated by many studies [14, 29, 72]. Among them, most similar to *minBlock* is optimal hashing [29], which also tries to reduce the size of 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 token-based strategy [121] as the only mechanism. The advantage of this strategy is to retain the highest number of the expected coreferences, compared to that of many other systems [62, 90, 112, 155], which use token weighting. Although weighting approach generates less number of candidates, it is accompanied by a considerable drop in recall.

5.3.5 Specification learning

The labeled candidates set C_L is separated into two sets, training set $Train$ and validation set Val. Using $Train$, Val, initial similarity functions F_{def} (Section 5.3.3), and similarity aggregators G_{inp} (Section 5.3.6), this step learns the optimal specification S_{opt} that is most suitable for the input repositories. A specification specifies the combination of similarity functions F, the similarity aggregator G, the parameters λ_δ associated with each similarity function δ, and the parameter ω of the *filtering* step (Section 5.3.7). The mission of this step is to automatically assign the optimal value to all elements of S_{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 specification 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 specification.
- $Init$ creates a specification by assigning G, F, and λ with given values.
- $Match$ executes the similarity aggregator and the *filtering* step in order to obtain the detected coreferent instances.
- $FindThreshold$ assigns a value to λ, a parameter of coreference filter. This function first selects the top $|Train^+|$ candidates with the highest matching score, where $|Train^+|$ is the number of the actual coreferences in $Train$. Then, it assigns the lowest score of the correctly detected coreference to λ.
- $Evaluate$ computes the performance of instance matching by comparing the generated results with the labeled data. In this function, $F1$ score is used as the

Algorithm 2: *cLearn*

Input: Training set $Train$, validation set Val, integer paramerter K_{top}
list of similiarity functions F_{def}, list of similiarity aggregators G_{inp}

Output: Optimal configuration S_{opt}

```
1  S_agg ← ∅
2  foreach A ∈ G_inp do
3      visited ← ∅
4      foreach δ ∈ F_def do
5          c ← Init(G ← A, F ← δ, λ ← 0)
6          links ← Match(c, Train)
7          c.λ ← FindThreshold(links, Train)
8          c.λ_δ ← c.λ
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(G ← A, F ← g.c.F ∪ h.c.F, λ ← 0)
17                 links ← Match(c, Train)
18                 c.λ ← FindThreshold(links, Train)
19                 F1 ← Evaluate(links, Train)
20                 visited ← visited ∪ {[c, F1]}
21                 if |g.c.F ∪ h.c.F| = |g.c.F| + 1 and g.c.F ≠ h.c.F 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     S_agg ← S_agg ∪ {[c, F1]}
26 [S_opt, F1] ← argmax_{c∈S_agg}(s.F1)
27 return S_opt
```

default performance metric. It is the harmonic mean of *recall* and *precision*, as calculated by Eq. 2.24 and 2.25.

- The validation set Val is used to increase the generality of the final specification C_{opt}. Each iteration controlled by line 2 finds an optimal specification with one similarity aggregator A. In other words, there are $|G_{inp}|$ specifications in S_{agg}. For selecting the most optimal one from S_{agg}, instead of just picking the specification 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, cross-validation is a reasonable option.

The heuristic used in this algorithm is the direct enhancement assumption (line 21). It states that the performance of using a combination must not be less than that of the combined items. This heuristic is reasonable as a list of similarity functions that reduces the performance has little possibility of generating a further list with improvement.

cLearn works under the principle of the well-known Apriori algorithm [1], 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. Similar to Apriori, *cLearn* works as a breadth-first search (BFS) algorithm and explores the search space by combining the previous states (i.e., the combinations of similarity functions). *cLearn* only combines the states of the same size s and share $s-1$ elements. Therefore, the size of the combination increases by one after each iteration. For example, in the 2^{nd} iteration, the combination of [1,2] and [1,3] (i.e., which makes [1,2,3]) is considered, but the combination of [1,2] and [3,4] (i.e., which makes [1,2,3,4]) is ignored. 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.

The global optimum of specification 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 [61, 100, 116] but this algorithm is still time-consuming because it is based on random convergence. In addition, genetic algorithm has many free parameters. *cLearn* uses a reasonable heuristic to reduce the complexity and minimize the parameters.

The complexity of Algorithm 2 is affected by many factors, including the size of training set $Train$, the number of initial similarity functions F_{def}, the number of similarity aggregators G_{inp}, parameter K_{top}, and the probability of the **if** statement in line 21. However, by analyzing only the number of the specification that the algorithm needs to

check, the size of training data is skipped. 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 is 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 the algorithm finish faster.

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.

5.3.6 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 specification. The computation of the matching score $mScore(x, y)$ for two instances x and y is defined as follows:

$$mScore_F(x, y) = \frac{1}{valid(U_F(x, y))} \sum\nolimits_{v \in U_F(x,y)} v^k \times weight(y) \tag{5.6}$$
$$U_F(x, y) = \{\delta(x, y) | sim(x, y) \geq \lambda_{sim}, \delta \in F\}$$

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:

$$weight(y) = \log_{\max_{t \in R_T} size(t)} size(y) \tag{5.7}$$

where $size(y)$ counts the number of RDF triples existing in y. By using Eq. 5.7, 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.

Equation 5.6 also reflects the generalization of different similarity aggregation functions described in Section 2.3.2. The difference is that it includes the weighting for target instance.

5.3.7 Filtering

This step produces the final coreferences. A candidate's coreferent possibility is not directly concluded from its matching score. Instead of that, the matching scores of all related candidates are considered. Similar to *ASL*, we reuse the principle of stable marriage problem [43]. The difference is that *ScLink* also uses a cut-off threshold λ to eliminate the incorrect candidates but satisfying Equation 4.6. λ is assigned automatically by the learning algorithm *cLearn*. Only candidates whose scores satisfy the above condition statement and threshold λ 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 coreferent if there exists other pairs (x, z) with higher similarity.

5.4 Experiment

5.4.1 Experiment target

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

TABLE 5.2: Summary of datasets used for *ScLink*.

ID	Name	$\|R_S\|$	$\|R_T\|$	P_S	P_T	$fact_S$	$fact_T$	$\|A\|$
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-GoogleProdu.	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

- 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 is allocated. 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.

5.4.2 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 5.2. 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. Datasets D9 to D15 are also used in previous experiments on $ScSLINT$ and ASL. We describe again those datasets in Table 5.2 for convenience.

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 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 have larger size. 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 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 large number of instances, the huge facts increase the ambiguity and complexity. Especially, the last 7 datasets (D9-D15) are realized as very challenging for theirs 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 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.

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 [37, 76, 150]. That enables the comparisons between *ScLink* and others systems.

5.4.3 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 for 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 will be generated. That quantity is comparatively high 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 whose type is string.

Table 5.3: Result of blocking using all property mappings.

ID	*size*	*pc*	ID	*size*	*pc*	ID	*size*	*pc*
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

For training data separation, we follow the workflow of *ScLink* (Figure 5.1). 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, 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}.

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

5.4.4 Experiment 1: Blocking

5.4.4.1 The general performance of *minBlock*

In this experiment, we compare the pair completeness 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 pair completeness (*pc*) for $C_0 = B_{def}(R_S \times R_T)$. Table 5.3 reports those results for all datasets. The high *pc* 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 pair completeness 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 [74], by using the advanced tri-gram attribute clustering [99]. The number of generated candidates is very small compared to all possible pair-wise instances between R_S and R_T. At least 99.9% unnecessary candidates are removed. However, compared to the actual coreferences, the generated candidates are generally still at much larger size.

TABLE 5.4: Cross validation result with *minBlock*.

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

The objective of *minBlock* is to remain the pair completeness 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 $C_U = B_{opt}(R_{SU} \times R_T)$.
- Finally, we calculate the two ratios: size reduction rate $rSize = 1 - \frac{|C_L|+|C_U|}{|C_{L0}|+|C_{U0}|}$ (larger is better) and pair completeness 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 pair completeness for unlabeled set because the pair completeness 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 pair completeness and the size of C_{L0} and C_{U0} are probabilistically similar to those of the parent set C_0 (reported in Table 5.3). Therefore, in order to enhance the readability, we skip the intermediate values and report only the ratios *rSize* and *rRec*.

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 pair completeness 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 5.4. 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, a considerable number of 1.26×10^9 candidates are discarded. These results confirm the effectiveness

and importance of *minBlock*. By limiting the candidates, the complexity of *cLearn* and *resolution phase* is much reduced.

5.4.4.2 Size of training data

For a supervised system, the size of training data is very important. Therefore, we also evaluate *minBlock* with different small amount 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. In 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. More detailed results are illustrated in Figure 5.2.

The reduction of $rRec$ is companied with the drop in $rSize$. However, the reduction of $rSize$ is not considerable and gradually slow down with the increase of labeled data. In 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 in acceptable range even we try to reduce $rRec$ by giving more labeled data to *minBlock*.

5.4.5 Experiment 2: Learning algorithms *cLearn*

5.4.5.1 General performance of *cLearn* and comparison to other 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 $F1$.

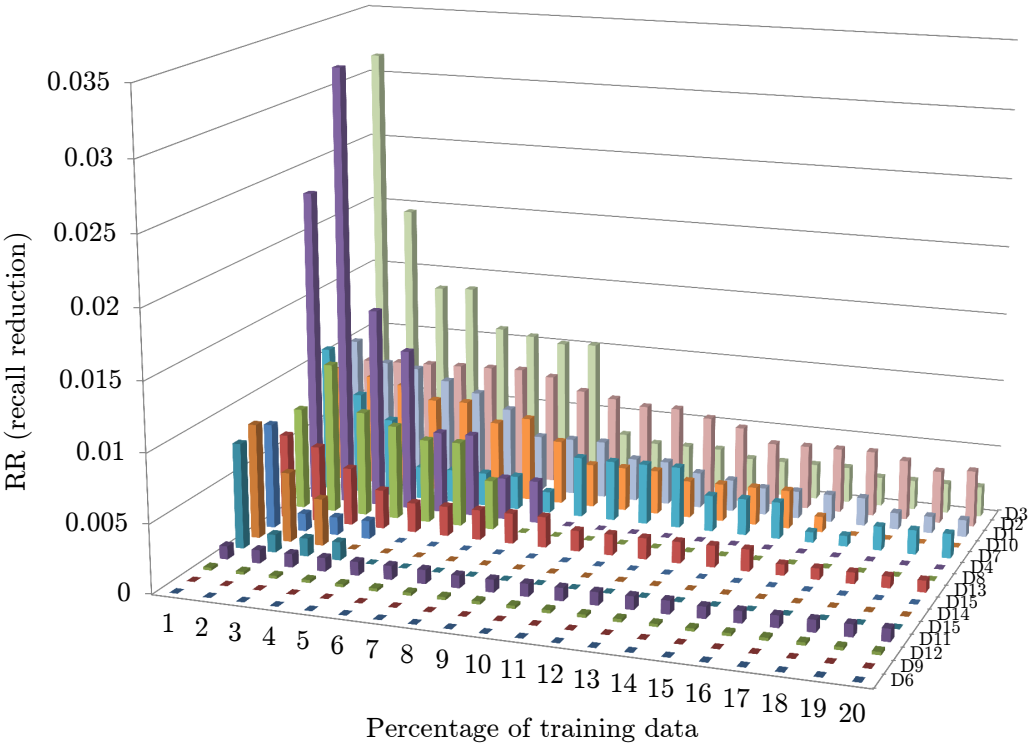


FIGURE 5.2: Pair completeness reduction rate by size of training data

- *gain* re-implements the idea of ADL [58], which selects the most discriminative property mappings by independently measuring the information gain of each property.

- *genetic* follows the idea of EAGLE [100], Knofuss [116], GenLink [60], and ActiveGenLink [61], which use genetic algorithm to learn the matching specification. We use binary array representation for the combination of similarity functions. We choose exponential ranking for fitness selection, 0.7 for single point cross-over probability, 0.1 for single point mutation probability, and 50 for the population size. 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 5.5: 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

Table 5.5 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 overall 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 specifications 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 specifications. Meanwhile, *genetic* needs to check 316 specifications in average. That is, the proposed heuristic reduces almost 44% the specifications while remaining the high accuracy, compared to *genetic*. In summary, it is concludable that *cLearn* is more reliable and faster than other tested algorithms.

5.4.5.2 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%.

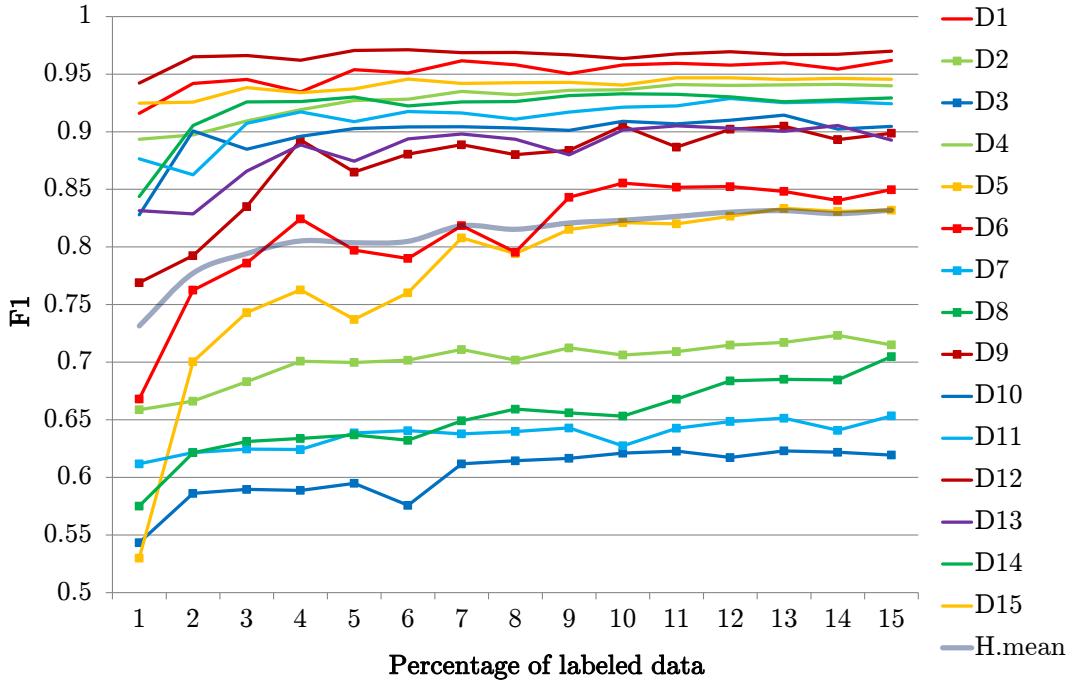


FIGURE 5.3: Performance of *ScLink* with different amount of labeled data.

The result of this experiment is depicted in Figure 5.3. 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 5.5). On most datasets, F1 score is 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%.

At 5% training data, *ScLink* expresses its capability by satisfying this expectation for most tested datasets. Although for few datasets, *ScLink* needs more than 5% labeled data to obtain the best F1, the optimal point that reconciles the performance and annotation effort is around 10% in overall. The results of the next experiment, which compares *ScLink* with other systems using small amount of labeled data, strongly support this conclusion.

5.4.6 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 specifications, we conduct some statistics on the specifications produced by learning algorithms. 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. The average number of learned similarity functions is about 8 for each test. 53% of the input similarity functions are removed by the learning process.

For similarity aggregator, considering each individual setting independently (e.g., $k = 1$ vs. $k = 2$ and *weighting* vs. *non-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 specifications. 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 *Modified-BM25*(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 particular input, it is difficult for a user to define a perfect matching specification like an automatic system can do.

For the evaluation of *Modified-BM25*, 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 5.6. According to this table, by including *Modified-BM25*, $ScSLINT$ improves the results for most datasets. Especially, for D10 to D14, which are among the large and highly ambiguous datasets, *using Modified-BM25*is clearly better. For D6, D8, D9, and D15, *using Modified-BM25*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, *Modified-BM25*shows its effectiveness against TF-IDF Cosine, especially for large and ambiguous datasets.

TABLE 5.6: F1 scores of *using* and *not using Modified-BM25*in *ScLink*.

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

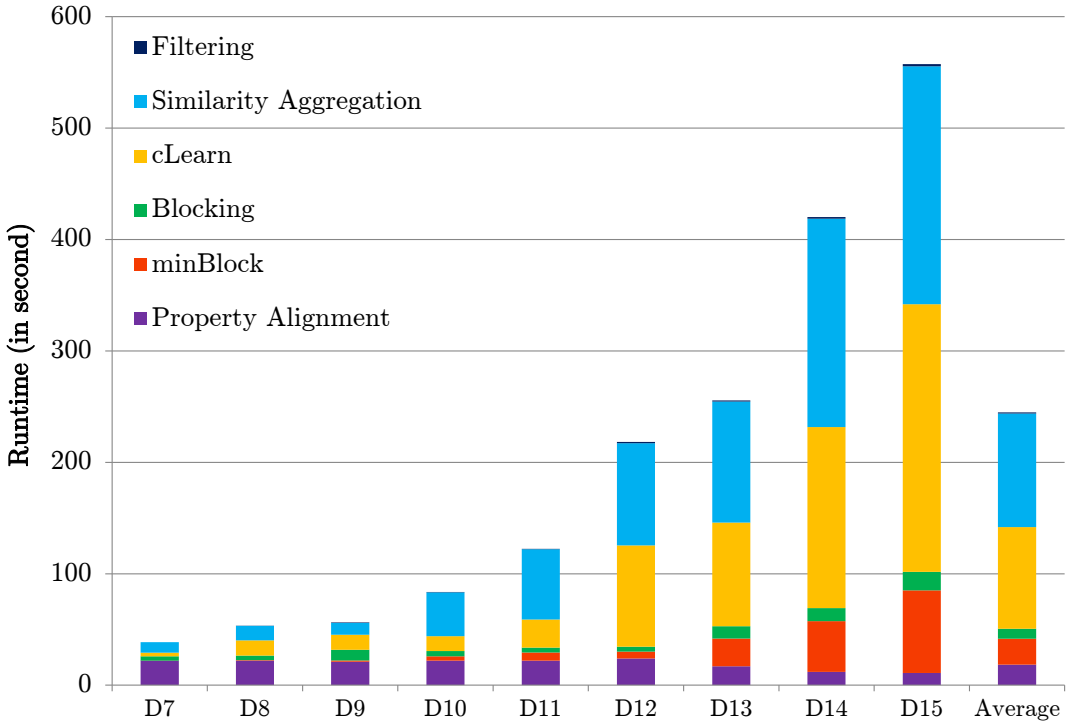


FIGURE 5.4: Detailed runtime of *ScLink* with 5% labeled data

5.4.7 Experiment 4: Runtime

Efficiency is an important factor of instance matching systems. Therefore, we evaluate the runtime of *ScLink* on tested datasets. 5% 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 steps, we plot the detailed runtime of the medium and large datasets in Figure 5.4. 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 deliver 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* [107], the total runtime for D15 is 36 minutes because the number of candidates is more than 10 times larger [107]. In summary, *ScLink* is very fast for large datasets and the consumed time of learning algorithms is also acceptable for the supervised scenario.

5.4.8 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 the result is also a criterion for selecting the systems. Since each system was tested on different datasets due to the support of data format and/or scalability, and the problem settings (e.g., supervised, non-supervised), we separate the comparison into three groups: D1 to D3, D4 to D8, and D9 to D15.

5.4.8.1 Datasets from 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 [22] and MARLIN [15], and the most recent work in [74], 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 5.7 reports the results of *ScLink* and these systems.

TABLE 5.7: 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

TABLE 5.8: 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

5.4.8.2 Datasets from D4 to D8

For these datasets, we compare *ScLink* with ObjectCoref [56] and the work in [134], 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 [80] and PARIS [151] 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 coreferences 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. We use the same amount of training data when comparing with each systems. Table 5.8 reports the comparison on these datasets.

5.4.8.3 Datasets from D9 to D15

Many systems have reported the experiments on these 7 datasets. Unfortunately, in 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 SERIMI [3] and ADL [58]; or only instances

TABLE 5.9: F1 scores of *ScLink* and other systems on D9 to D15.

	ScLink (5%)	*ScLink* (10%)	Zhishi.Links	AggrementMaker	ASL
D9	0.87	0.90	**0.91**	0.85	0.88
D10	0.90	0.91	**0.92**	0.69	0.88
D11	0.91	**0.93**	0.91	0.74	0.87
D12	**0.97**	**0.97**	**0.97**	0.88	0.96
D13	0.88	**0.90**	0.88	0.85	0.90
D14	0.93	**0.94**	0.87	0.80	0.92
D15	0.94	0.94	0.93	**0.96**	0.93
H.mean	0.913	**0.925**	0.912	0.816	0.904

residing in actual coreference (R) are used by ADL [58], Knofuss [116], ActiveGenLink [61], and SLINT+ [112]. As using such simple datasets, it is not difficult to obtain the perfect results using simple methods, as reported in by SLINT+ [112] and SERIMI [4].

For reference, we report the comparisons with Zhishi.Links [117] and AgreementMaker [27], and *ASL*, which are tested with full datasets though they are not learning-based systems. We report the result of using 5% and 10% of labeled data for the learning algorithms because 10% is popularly used as an expectation for the size of labeled data. Table 5.9 reports this comparison.

In over all, *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 coreferences inside the source or target repository. *ScLink* clearly outperforms AgreementMaker 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 systems 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.

5.5 Summary

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 specification and blocking model. We also use a robust string similarity metric. We reported the detail analyzes 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 specification learning. Together with effectiveness, *ScLink* is also more efficient in both time and memory consumption.

Chapter 6

R2M: Ranking features for classification-based instance matching

In this chapter, we present the *R2M* ranking feature for classification-based instance matching. We describe the motivation (Section 6.1) and the problem statement (Section 6.2). We describe the background of classification-based instance matching and the space of ranking feature inclusion (Section 6.3). After that, we describe the *R2M* feature (Section 6.4). In the last section (Section 6.5), we report the experiment.

6.1 Motivation

Classification is the problem of predicting the category (i.e., class) of unseen examples, on the basis of a training data containing the seen examples, whose categories are observable. Classification is a form of supervised method and is supported by the solid machine learning theory. Classification covers a wide range of applications, mainly for prediction tasks.

Instance matching can be considered as a classification problem in which each example represents a pair of instances. The information described by an instance include a feature vector consisting of correlation indicators (e.g., literal similarities). For training data, an instance is also associated with a class, which is either coreferent (i.e., positive) or non-coreferent (i.e., negative). The matching task on a new example is to accurately predict its actual class.

An important technique of instance matching is the ranking of matching results. For easier description, we can consider the instance matching as the problem of information retrieval. Given an instance x in the source repository, the goal is to find in the target repository a list of instances $Y = \{y\}$ so that every pair $\langle x, y\rangle$ is coreferent. That is, among all instances of target repository, we have to rank and select the best-matched instances to construct Y. The key difference of instance matching is that it is not necessary to always exist a coreferent instance of x. In other words, Y can be empty. In contrast, traditional information retrieval is more constraint-relax. The task is to collect the resources that are most similar without determining whether they are truly matched or not.

Ranking is important in instance matching. The main reason is that different pairs of $\langle x, Y\rangle$ (i.e.,$\langle x_1, Y_1\rangle$, $\langle x_2, Y_2\rangle$, ...) have different level of ambiguity. That is because each pair of $\langle x, Y\rangle$ is constructed with unique values (e.g., blocking key, token) and the ambiguity is closely related to the values. For example, suppose that $\langle x_1, Y_1\rangle$ shares the token Tokyo and $\langle x_2, Y_2\rangle$ share the token Chiyoda. Now, because there are many instances describe about entity related to Tokyo (e.g., Tokyo metropolis, Tokyo Skytree, and Tokyo station) than Chiyoda (e.g., Chiyoda ward and Chiyoda corporation), it is more difficult to discriminate the coreferences and non-coreferences among all pairs of $\langle x_1, Y_1\rangle$ than those of $\langle x_2, Y_2\rangle$. Traditional classification model tends to discriminate the positive and negative based on the major cases of the data. That is, the boundary to separate the two classes are fixed for all examples of different ambiguity tiers. That manner is potentially problematic for the data with high variance of ambiguity. Therefore, is it important to include the ranking of examples in classification-based instance matching.

In *ScLink* and *ASL*, stable matching has demonstrated the effectiveness. Stable matching is a form of ranking in which only the candidates of highest matching score among all of $\langle x, Y \rangle$ are qualified. It implies a possibility of improvement when applying ranking techniques to classification-based instance matching. Furthermore, *ScLink* and *ASL* are unable to measure the impact of the ranking factor, even when *ScLink* is a supervised system. In fact, the ranking factor is varied for different repositories as the ambiguities are different. Using ranking factor as a feature of training examples is helpful because the learning algorithm is capable of determining the impact of the ranking factor.

Previous effort on using classification algorithms for instance matching demonstrated promising performance [20, 81, 114, 134, 143]. However, the inclusion of ranking mechanism remains unsolved and that leaves a possibility of improvement. We are motivated from the employment of ranking techniques for classifier in order to simultaneously predicting the coreference and enhancing the disambiguation capability.

6.2 Problem statement

The problem of supervised instance matching is defined as in Section 5.2. Given two repositories the source R_S and the target R_T, the goal is to identify the set I of coreferences. In this work, we focus on a more specific problem. That is the learning of a classification model with ranking factor. Therefore, the problem is detailed as follows.

- Given a set of examples, each example represents the correlation of two instances $\langle x, y \rangle$, one is from R_S and the other is from R_T.
  - A typical form of correlation is the literal similarities of x and y. In this chapter, we use the term *example* to mention the feature vector. For example, $f(a)$ is to apply a function f on the feature vector of example a.
  - The set of examples can be obtained by taking the pairwise instances of R_S and R_T or inheriting the candidates generated by blocking techniques.
- The set of examples are divided into two sets, training and test set.
  - Training set contains the curated examples whose the coreferent status is known. In other words, the class (positive/negative) of these examples are given.
  - Test set contains the examples whose class information is missing.
- The problem is to learn a classifier $\mathcal{M}$ from the training data and to use $\mathcal{M}$ to predict the class $\mathcal{M}(a) = \{1, 0\}$ of each example a of the test set, where 1 and 0

TABLE 6.1: The notations used in classification-based instance matching

Symbol	Meaning
a, b, c	examples (i.e., feature vectors)
$\mathcal{X}$	set of examples
$\ell(a)$	class of example a
a^*	examples related to a (i.e., the block to which a belongs)
f	ranking function
w	parameters of ranking function (i.e., weight vector)
$\mathcal{M}$	classifier model
$\mathcal{G}$	preference function
L	loss function

stand for positive and negative, respectively. In other words, $\mathcal{M}$ can be considered as a mapping function, $\mathcal{M}$: $\mathcal{X} \rightarrow \{1, 0\}$, where $\mathcal{X}$ is the set of examples.

6.3 Preliminary

This section provides the general picture of classification-based instance matching. We also describe how a ranking factor can be included in the traditional classifier. Before getting to the detail, we note the frequently used symbols, which are listed in Table 6.1.

6.3.1 Classification-based instance matching

In classification-based instance matching, the input for learning algorithm as well as for prediction task is the examples representing the correlation of instances. To our best knowledge, the correlation reflects the similarity of the instances. Such similarity can be the literal similarities or the final matching score. Because the literal similarities provide more detail information so that the learning algorithm can find the optimal classifier, literal similarities are widely used to construct the examples. Based on the observation of existing classification-based instance matching systems, we illustrate a typical workflow that matches to all systems. Figure 6.1 depicts the workflow.

According to Figure 6.1, there are six components in the workflow.

- The first component is property alignment, which creates the property mappings. This component is automatically or manually operated.

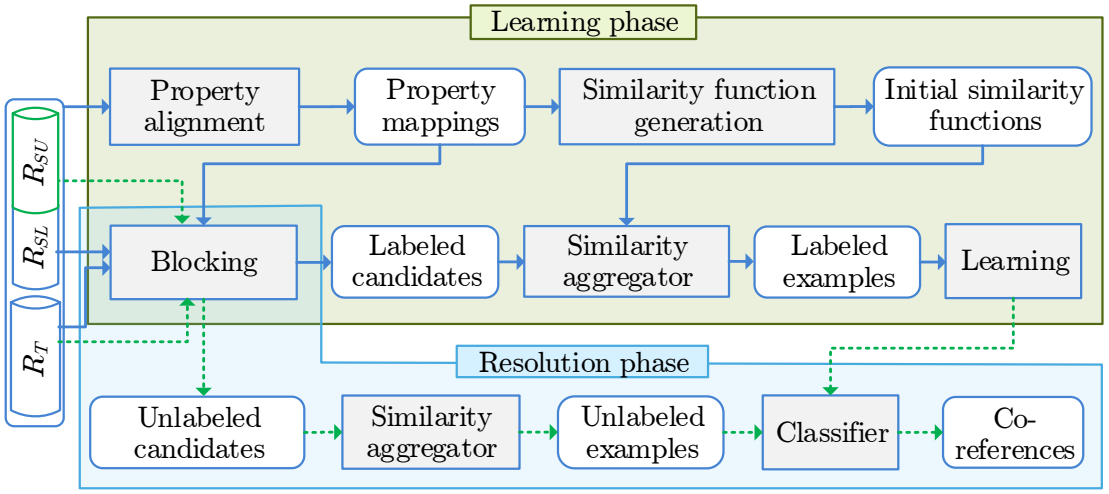


FIGURE 6.1: A typical workflow of classification-based matching.

- The second component, similarity function generation, assigns the similarity metrics to each property mappings and creates the initial similarity functions, which are later used to estimate the literal similarities.
- The third component is the blocking. This component generates the candidates, which are the pairs of instances having a possibility to be coreferent. The candidates are divided into two parts, labeled and unlabeled sets. The labeled candidates are used to create the training data while to detect the coreferences among unlabeled candidates is the mission of the system.
- Using the initial similarity function generated in the second component, the fourth component, similarity aggregator, computes the literal similarities for all candidates and generates the examples for classification. The labeled and unlabeled candidates are used to construct the training and test data, respectively.
- Training data is input to the learning component to train a classifier.
- Finally, the last step is to apply the classifier on unlabeled examples, to predict their matching status.

As we earlier mentioned in Section 6.2, the examples can be the pairwise instances of R_S and R_T. However, the blocking step is installed in this workflow for generalization. If a system uses all pairwise instances, the blocking step simply returns all of them without any filter.

The above workflow is very similar to the first 4 steps of *ScLink*. The key differences are the *learning* and the *classifier*. Classifier can be any model with the function that maps

each unseen example into the class positive or negative. Classification is a extensively studied area and therefore, plenty of classifiers are available. Some examples of the classifier for instance matching are J48 decision tree, Support vector machine (SVM), Random forest, and Multilayer perception, which have shown remarkable performances [74, 81, 114, 134].

6.3.2 Ranking

In machine learning, the ranking is the problem of predicting the order of examples. For example, to predict the preference of an example among two or a larger collection of examples. This task is named learning to rank. As reviewed in Section 2.6.7, learning to rank consists of three approaches: pointwise, pairwise, and listwise. Among them, pairwise ranking is the most suitable form for instance matching. This approach induces the ranking of examples by considering the relationship of every possible pairs of instances.

Pointwise approach considers the ranking of each example independently, it is not suitable for our goal in instance matching. Listwise approach finds the preference of an example over a sorted collection. In instance matching, a sorted collection like that is not available because only positive/negative labels are observable. Concretely, given the set $\langle a, B\rangle$, if a pair $\langle a, b_i\rangle$ is positive and the others are negative, the preference can be defined only for each of $[\langle a, b_i\rangle, \langle a, b_j\rangle]$, rather than for all of $[\langle a, b_j\rangle, \langle a, b_k\rangle]$, where $i \neq j \neq k$. Shortly, if we can only make sure that the first ranked example is correct, the order of the remaining examples is left unknown. Therefore, it is not suitable to apply listwise approach as well.

The general idea of pairwise approach is to find a ranking function f so that the order of two examples a and b can be defined by $sign(f(a) - f(b)) = \{1, -1\}$, where 1 means a has higher preference than b and -1 means otherwise. f can be represented in different forms. A typical form f is the linear combination of the input feature vector: $f = w \times x$, where w is called parameters of f. Another common form of f is the logistic function $f = \frac{1}{1+e^{-wx}}$. The performance of the ranking is determined by the parameter w and tuning w using training data is the task of learning algorithm. Basically, the learning algorithm optimizes w by minimizing the following loss:

$$L(f, w, \mathcal{X}) = \sum_{a \in \mathcal{X}} \sum_{b \in \mathcal{X}, \ell(b) < \ell(a)} \mathcal{G}(f(a) - f(b)) \tag{6.1}$$

where $\mathcal{G}$ is a preference function. $\mathcal{G}$ can be a hinge function $\mathcal{G}(z) = (1 - z)_+$, exponential function $\mathcal{G}(z) = e^{-z}$, or logistic function $\mathcal{G}(z) = \frac{1}{1+e^{-z}}$. In order to simply minimize L, $\mathcal{G}$ is usually chosen so that L is convex (e.g., above listed form of $\mathcal{G}$). In that case, the

global minimal of L is possible and identifiable. The detail of optimization algorithm is given in Section 6.4.2 in the context of our proposed ranking feature *R2M*.

6.3.3 Combination of classification and ranking

It is possible to include the ranking factor in the classifier. To our best knowledge, CRR [142] is the only one work that focuses on the combination of classification and ranking. The general idea of CRR is to produce a model that can score the examples so that the result can help to make the class prediction as well as is capable of ranking the examples. The detail of this combination is as follows.

$$L^{CRR}(f, w, \mathcal{X}) = \alpha L^R(f, w, \mathcal{X}) + (1 - \alpha)L(f, w, \mathcal{X}) \tag{6.2}$$

where $L(f, \mathcal{X})$ is the ranking loss (Equation 6.1) and $L^R(h, \mathcal{X})$ is the classification loss, which is the logistic regression:

$$L^R(f, w, \mathcal{X}) = \frac{1}{|\mathcal{X}|} \sum_{z \in \mathcal{X}} h(\ell(z), f(z)) \tag{6.3}$$

where:

$$h(y, f(z)) = y \times f(z) + (1 - y) \times (1 - f(z)) \tag{6.4}$$

and f is a logistic function $f(z) = \frac{1}{1+e^{-z}}$.

The advantage of this method is that the learning process can simultaneously optimize the classifier and ranking factor. In addition, the impact of regression and ranking are controllable by adjusting the meta-parameter $\alpha \in [0, 1]$. The priority of regression is proportional to the increase of α. However, CRR still contains some drawback. First, the trained model f has to carry two tasks regression and ranking, but is configured by only one parameter w. It is potentially ineffective as the ideal models of regression and ranking may be very different. Second, it is difficult to extend the idea to combine ranking and another classifier. The training of function-based classifier (e.g., Logistic regression, SVM, and Neural Network) can be done by minimizing the loss function like Equation 6.3. However, for some type of models, for example tree-based (e.g., ID3, and J48) or rule-based (e.g. IDA and OneR), it is impossible to include the ranking into the trained model using the 1 linear combination like CRR.

In our work, we try to include the ranking factor in the manner that is independent of the mechanism of the classifiers. We propose to include the ranking factor as elements of the feature vector. That is, the values derived from ranking function $f(x)$ can be

included in the original feature vector x to create a new vector x'. Similarly, given a set of examples $\mathcal{X}$, another new set of examples is created and the classification problem is deployed on the new data $\mathcal{X}$'. More detailed, the learning process will contain two steps. The first step is to learn the ranking function f from the original data $\mathcal{X}$ and the second step is to learn the final classifier $\mathcal{M}$ from the modified data $\mathcal{X}' = \mathcal{X} \cup f(\mathcal{X})$.

The advantage of this method is that the classification model $\mathcal{M}$ is independent with the ranking model f. That is, the ranking model is compatible with any classifier so that it is possible to employ any classifier. Furthermore, because the parameters of classifier $\mathcal{M}$ and those of ranking model f are independent, $\mathcal{M}$ and f can better adapt to the characteristic of the data.

6.4 *R2M* ranking feature

In this section, we describe the proposed *R2M* ranking feature [110] and how to learn the ranking model.

6.4.1 Feature design

Recall that an example represents the correlation of two instances. Let a is the example of the pair $\langle x, y \rangle$, we denote $a*$ as the set of all examples representing all the $\langle x, z \rangle$ related to x, where $z \neq y$. An example of $\langle x, z \rangle$ is to take the result of blocking step, when $\langle x, z \rangle$ is in the same block. The general idea of *R2M* is to reflect the preference of an example over other related examples. It takes the difference between the number of cases a has more preference than the cases a has less preference. This idea can be formulated as follows.

$$R2M_0(a) = |\{b|f(a) > f(b), b \in a^*\}| - |\{b|f(a) < f(b), b \in a^*\}| \tag{6.5}$$

which is equivalent to:

$$R2M_0(a) = \sum_{b \in a^*} \text{sign}(f(a) - f(b)) = \text{sign}(\mathcal{G}(a, b)) \tag{6.6}$$

where $\mathcal{G}(a, b) = f(a) - f(b)$ is the preference function and f is the ranking function.

The above function is quantitative because it measure the number of preferences and ignores the difference between cases having the same preferences. Therefore, we finally design a feature that can carry the qualitative reflection. In addition, the normalization

is necessary. The *R2M* feature of an example a is defined as follows.

$$R2M(a) = \frac{1}{|a^*|} \sum_{b \in a^*} \mathcal{G}(a, b) = \frac{1}{|a^*|} \sum_{b \in a^*} (f(a) - f(b)) \tag{6.7}$$

We use logistic for ranking function f.

$$f(c) = \frac{1}{1 + e^{-wc}} \tag{6.8}$$

where w is the weight vector, the parameter of f and need to be optimized by learning algorithm. The logistic function is widely used in classification and regression because it has a good ability to normalize the input value into the range of $(0, 1)$, which is useful in analyses. Furthermore, logistic function is convex and easier to be optimized. Next, we describe the learning of parameter w.

6.4.2 Optimization

The task of optimization is to find the parameter w given a set of training examples $\mathcal{X}$. Gradient descent is a useful algorithm to optimize the ranking function. In order to apply the gradient descent, we first define the loss function for the ranking function f over the training set $\mathcal{X}$. After that, we optimize the loss function using gradient descent method. The loss of a training example is defined as the complement of the *R2M* feature.

$$L(f, w, a) = -\sum_{b \in a^*} \mathcal{G}(a, b) \tag{6.9}$$

which is equivalent to

$$L(f, w, a) = \sum_{b \in a^*} \mathcal{G}(b, a) \tag{6.10}$$

where a^* is the examples related to a. Furthermore, in the learning process, only the example pairs of different classes are considered. Let

$$a^- = \{b | \ell(b) < \ell(a)\} \tag{6.11}$$

We consider only those pairs because the $\ell(a)$ and $\ell(b)$ are same if a and b belong to the sane class. Moreover, considering the cases that $\{b|\ell(b) > \ell(a)\}$ is not necessary because it causes the redundancy as $\mathcal{G}$ is a symmetric function: $\mathcal{G}(a, b) = -\mathcal{G}(b, a)$.

From the loss of each example, we induce the loss over the training set $\mathcal{X}$, which is the sum of loss of all examples in $\mathcal{X}$.

$$L(f, w, \mathcal{X}) = \sum_{a \in \mathcal{X}} \sum_{b \in a^-} \mathcal{G}(b, a) \tag{6.12}$$

In order to normalize the loss function, the average is applied to the loss of each example and also to all examples. That is the above loss is refined as follows.

$$L(f, w, \mathcal{X}) = \frac{1}{|\mathcal{X}|} \sum_{a \in \mathcal{X}} \sum_{b \in a^*} \frac{\mathcal{G}(b, a)}{|a^-|} \tag{6.13}$$

Finally, we the regularization factor is introduced to prevent the over-fitting, as similar to other loss function optimization. Here we use the L2-norm regularization.

$$L(f, w, \mathcal{X}) = \frac{1}{|\mathcal{X}|} \Big(\sum_{a \in \mathcal{X}} \sum_{b \in a^-} \frac{f(b) - f(a)}{|a^-|} + \frac{1}{2} \lambda ||w||^2 \Big) \tag{6.14}$$

where $\lambda \geq 0$ is the regularization parameter. Larger value of λ puts more regularization. In other words, larger value of λ helps prevent more over-fitting and smaller value of λ helps prevent more under-fitting. In practice, the regularization parameter λ is determined by the learning process, using validation data.

The gradient descent is an iterative method that minimizes the loss by estimating the local minimal of $L(f, w, \mathcal{X})$. It works on the basis of finding the parameter w that makes the gradient of $L(f, w, \mathcal{X})$ close to zero, by following the Lagrange multipliers. Gradient descent updates the parameter w into the new one w' at each iteration by the following rule.

$$w' = w - \alpha \frac{\partial L(f, w, \mathcal{X})}{\partial w} \tag{6.15}$$

where α is the learning rate and $\frac{\partial L(f,w,\mathcal{X})}{\partial w}$ is the gradient of L with respective to w as is calculated as follows.

$$\frac{\partial L(f, w, \mathcal{X})}{\partial w} = \frac{1}{|\mathcal{X}|} \Big(\sum_{a \in \mathcal{X}} \sum_{b \in a^-} \frac{(\frac{\partial f(b)}{\partial w} - \frac{\partial f(a)}{\partial w})}{|a^-|} + \lambda w \Big) \tag{6.16}$$

where the gradient of logistic function f is as follows.

$$\frac{\partial f(c)}{\partial w} = f(c) \times (1 - f(c)) \times c \tag{6.17}$$

The gradient descent algorithm is summarized as the following pseudo-code.

Algorithm 3: Gradient descent

Input: Training set $\mathcal{X}$, regularization parameter λ, learning rate α

Output: Trained parameter w

```
Initialize w as ones vector
repeat
    Calculate loss L by 6.16
    Calculate gradient of L by 6.18
    Update w by 6.17
until convergence
return w
```

According to Algorithm 3, the learning process finishes when the convergence of L is reached. The algorithm returns the parameter w and it is used to calculate the *R2M* feature for unseen examples.

6.5 Experiment

6.5.1 Experiment settings

The purpose of experiment is to validate the effect of *R2M* on the classification-based instance matching systems. For doing that, we design a workflow that inherits parts of *ScLink* for literal similarity estimation and includes the new elements for classification-based instance matching. The concrete workflow is as follows.

- Given the input repositories, we first apply the learning phase of *ScLink* with skipping the specification learning step (Figure 5.1). That is, we reuse the property alignment, blocking (including the learning algorithm *minBlock*), and similarity function generation. The results include the sets of labeled candidates and unlabeled candidates, and the initial similarity functions.
- We apply all initial similarity functions on all candidates and record the feature vectors. We construct the training data and test data with respective to labeled and unlabeled candidates.
- We apply the learning algorithm (Section 6.4.2) to learn the *R2M* model on labeled candidates. Then, we apply the trained model to compute the *R2M* feature of each candidate, including labeled and unlabeled ones. This feature is merged with the original features to construct new examples with extra ranking feature. In addition,

the training set is divided into two sets, one of them is used as the validation set to learn the regularization parameter w.

- We apply the optimization algorithm to train a classifier on those new examples.
- We apply the classifier on test data to predict the label of each test example. The results are compared to the ground-truth for making the final evaluation result. The metrics used for this experiment are precision, recall, and F1 scores.

We conduct in total 2 experiment. The first experiment is to see the performance of *R2M*. For doing that, we compare the results of the classifier when using the original features and when using *R2M*. In addition, we also take the results of *ScLink* into the comparison. The second experiment is to analyze the trend of performance when varying the size of training data. For each experiment, we use four state-of-the-art classifiers, including Logistic regression, J48, SVM, and Random forest. Before reporting the experiment results, we describe the datasets.

6.5.2 Datasets

That is, for each dataset and each classification algorithm, the classifier trained on original feature vectors is compared to the one trained on modified feature vectors. For each dataset, we conduct two splitting strategies for training and test data. Concretely, we employ 5 folds cross validation and percentage split. The cross validation is used to know the performance of *R2M* on the stable training data. Meanwhile, the percentage split is used to see the trend of performance when giving different effort of curation. We vary the amount of training data from 2% to 20%. We reduce the random noise by percentage splitting, we repeat the experiment 10 times and measure the average results.

In summary, there are 1560 tests need to be done for each dataset. That number of tests is large and therefore it is sufficient to conduct the experiment on a limited number of datasets. We use 8 datasets for this experiment. They are the first 8 of 15 datasets used for testing *ScLink*. Table 6.2 is the summary of the datasets. In this table, the number of examples is calculated by taking the result of applying *minBlock* with 5-folds cross-validation. For later experiments, we adjust the size of training data and the number of examples will be varied accordingly. The number of features are fixed for all experiments because the result of property alignment is based on all data of input repositories. The number of positive examples are identical to the number of expected co-references. The number of negative examples are the complement of positive examples, in the parent set all examples.

Table 6.2: Summary of datasets used for testing *R2M*.

ID	Name	#Examples	#Feature	#Positive
D1	DBLP-ACM	341,446	37	2,224
D2	ABT-Buy	61,756	23	1,097
D3	Amazon-GoogleProduct	70,550	27	1,300
D4	Sider-Drugbank	5,362	59	1,142
D5	Sider-Diseasome	4,227	31	344
D6	Sider-DailyMed	5,013	31	3,225
D7	Sider-DBpedia	468,909	58	1,449
D8	Dailymed-DBpedia	951,135	114	2,454

Table 6.3: F1 scores of using and not using *R2M* with 5 folds cross-validation

ID	LR		J48		RR		SVM	
	Origin	*R2M*	Origin	*R2M*	Origin	*R2M*	Origin	*R2M*
D1	0.9702	*0.9774*	0.9736	*0.9866*	0.9850	***0.9886***	0.9652	*0.9748*
D2	0.5956	*0.5980*	0.5766	*0.6028*	0.6470	***0.6736***	0.5410	*0.6024*
D3	*0.5508*	0.5224	0.5252	*0.5682*	0.5766	***0.6380***	0.4204	*0.5114*
D4	0.9480	*0.9710*	0.9564	*0.9740*	0.9704	***0.9810***	0.9486	*0.9734*
D5	*0.9148*	0.9114	0.9036	*0.9076*	*0.9472*	***0.9472***	0.9436	*0.9436*
D6	*0.9548*	0.9536	0.9780	*0.9854*	0.9804	***0.9902***	0.9578	*0.9586*
D7	*0.7104*	0.7046	***0.7258***	0.7232	0.6886	*0.6994*	0.7096	*0.7156*
D8	0.7934	*0.7968*	0.8258	*0.8398*	0.8562	***0.8638***	0.7930	*0.8094*

6.5.3 General performance of *R2M*

In this experiment, we compare the performance of the classifier when using and not using *R2M* feature. We use 5 folds cross-validation for this experiment in order to know the stable performance of the classifier as well as *R2M*. Table 6.3 reports the F1 scores of 4 classifiers: Logistic regression (LR), J48 decision tree, Random Forest (RR), and Support Vector Machine (SVM). In this table, Origin and *R2M* represent that the *R2M* feature is not used or used, respectively. The italic and bold numbers indicate the best result in the context of same classifier and dataset, respectively. According to this table, using *R2M* enhances the performance in 26 out of 32 tests (81%). Especially when using a classifier other than LR, *R2M* does not make effect on at most one dataset. Although *R2M* reduces the performance on few datasets, the difference is not remarkable. Furthermore, the improvement when using *R2M* compared to using only the original features is very statistically significant, according to a paired t-test (p=0.0012).

We also compare classification-based matching using *R2M* and the specification-based matching system *ScLink*. The result of *ScLink* when using 5 folds cross-validation is given in Table 6.4. Compared to *ScLink*, classification-based matching is generally better on all datasets except D2 and D3. Consider only the best performed classifier

TABLE 6.4: F1 scores of *ScLink* and classifier RR with 5 folds cross-validation

ID	*ScLink*	RR-Origin	RR-*R2M*
D1	0.9626	0.9850	**0.9886**
D2	**0.6918**	0.6470	0.6736
D3	0.6102	0.5766	**0.6380**
D4	0.9486	0.9704	**0.9810**
D5	0.8536	**0.9472**	**0.9472**
D6	0.8630	0.9804	**0.9902**
D7	0.6591	0.6886	**0.6994**
D8	0.7316	0.8562	**0.8638**

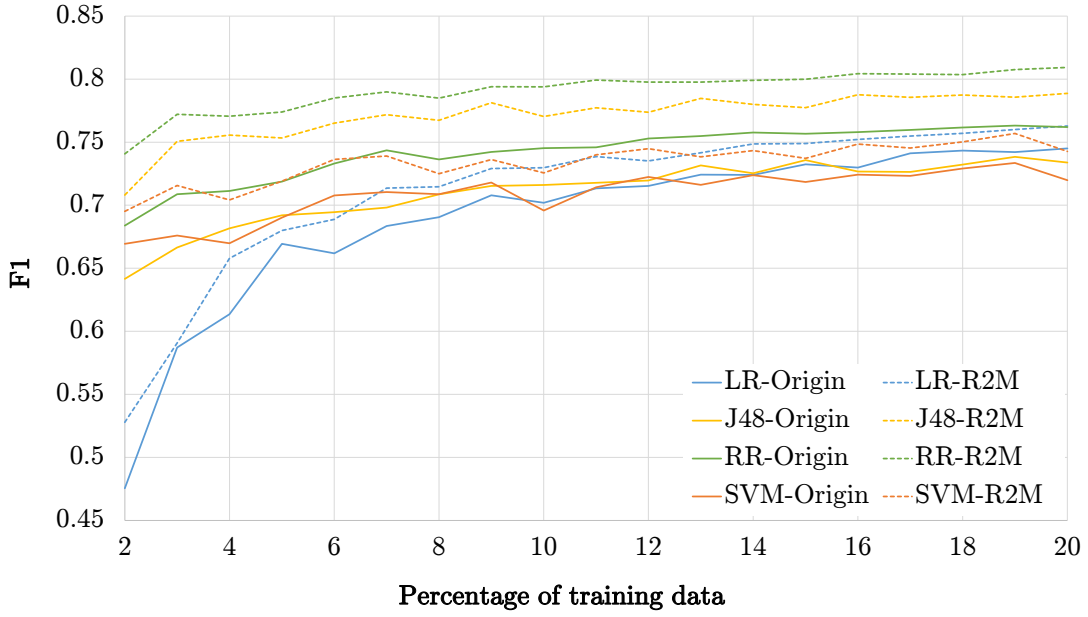


FIGURE 6.2: Harmonic mean of F1 scores by curation effort.

Random forest, an interesting result is that using *R2M* can significantly improves the F1 score (p=0.0191) but this such result is not obtained when not using *R2M*.

The above results confirm the role of ranking factor in classification-based instance matching. It also reveals the effectiveness of our proposed *R2M* feature on real datasets.

This experiment uses 5 folds cross-validation to better guarantee the stability of training data (80%). However, in practice, it is prioritized to use a smaller amount of training data in order to reduce the curation effort. In the next experiment, we analyze the variation of performance when changing the size of training data.

6.5.4 Size of training data

We analyze the movement of F1 scores when changing the amount of training data. We vary the training set from 2% to 20%, the remaining data is used for testing. For

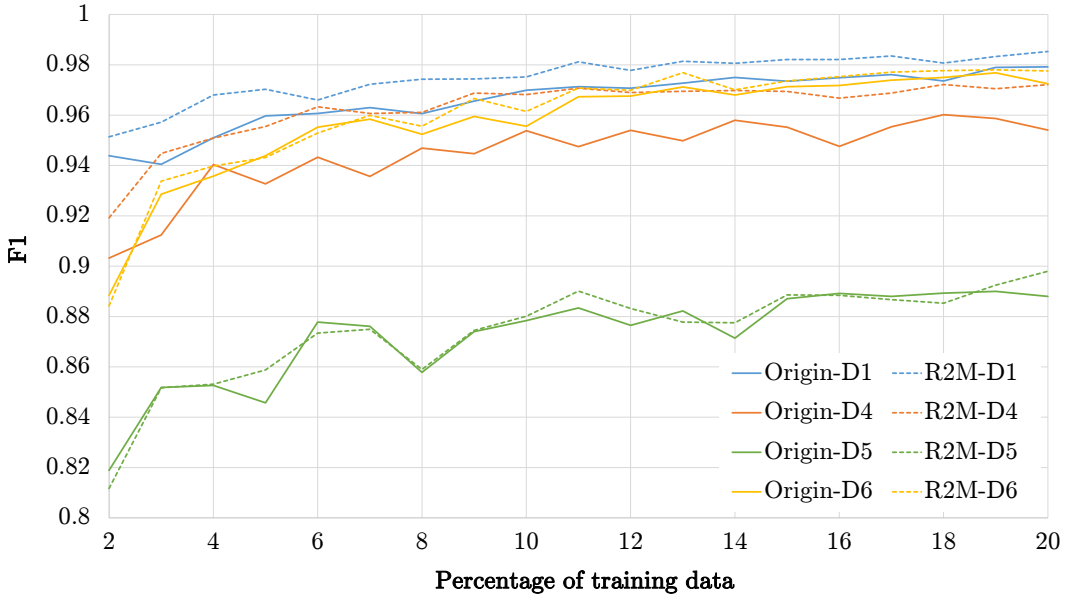


FIGURE 6.3: F1 scores by curation effort using Random Forest on D1, D4, D5, D6.

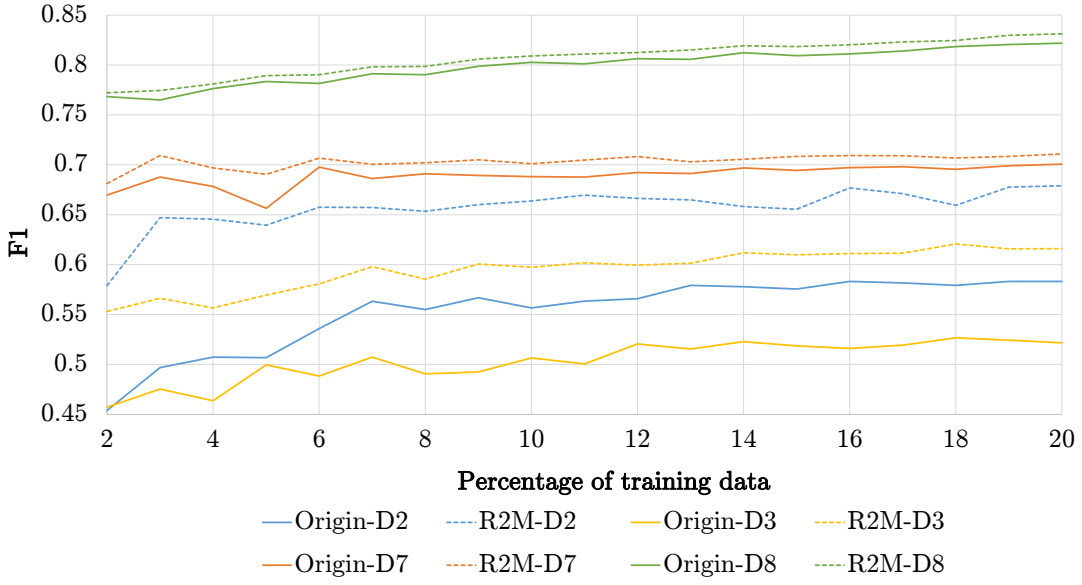


FIGURE 6.4: F1 scores by curation effort using Random Forest on D2, D3, D7, D8.

each split setting, we repeat the test 10 times in order to reduce the random noise. For reporting the result of each algorithm, we take the harmonic mean of F1 scores over all datasets. This result is illustrated in Figure 6.2. Considering the best performed classifier, we report the detailed result of each dataset using Random forest. The results are split into two graphs in accordance to the similarity of F1 scores. Figure 6.3 reports the result on D1, D4, D5, and D6. Meanwhile, Figure 6.4 includes the result on D2, D3, D7, and D8.

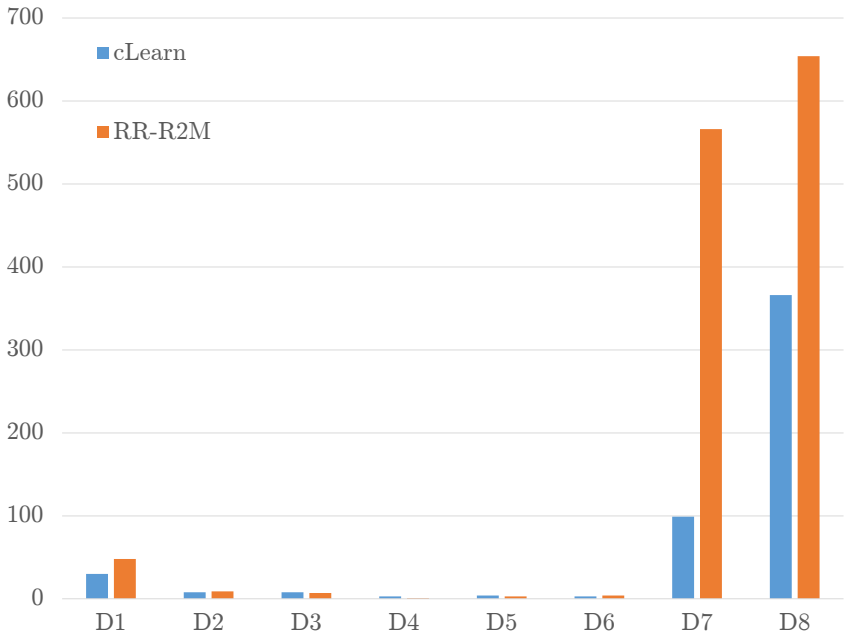


FIGURE 6.5: Training time of *cLearn* and *R2M* + Random forest (in second).

According to these figures, generally more amount of training data delivers better performance. The harmonic mean of F1 scores quickly increases when adding more training data within the first 7%. After this point, the performance go up slowly. An interesting result is that at 10% the difference between the results and those at 5 folds cross-validation, which employs 80% of training data, is not statistical significant (p=0.0129). Therefore, it is reasonable to conclude that a limited amount of training data is sufficient to train a good classifier for the instance matching system. Moreover, the similar result is also obtain in the experiment on *ScLink*, the specification-based matching system. The experiments reveal that supervised instance matching is practical because of the minimal requirement of curation effort.

6.5.5 Discussion

A limitation of classification-based instance matching is the scalability issue. For supervised specification-based, in order to obtain a good specification, a medium to large number of initial similarity functions should be input into the learning process. However, in resolution phase, not all initial similarity functions is necessary. The learned specification contains only the useful similarity functions, which are up to 53% reduced (Section 5.4.6). For classification-based matching, because the difficulty of model interpretation, all similarity functions are required for the unlabeled candidates to construct the examples of the same structure with the training data. As a literal comparison, the average runtime of *ScLink* and RR-*R2M* (including training and prediction), when

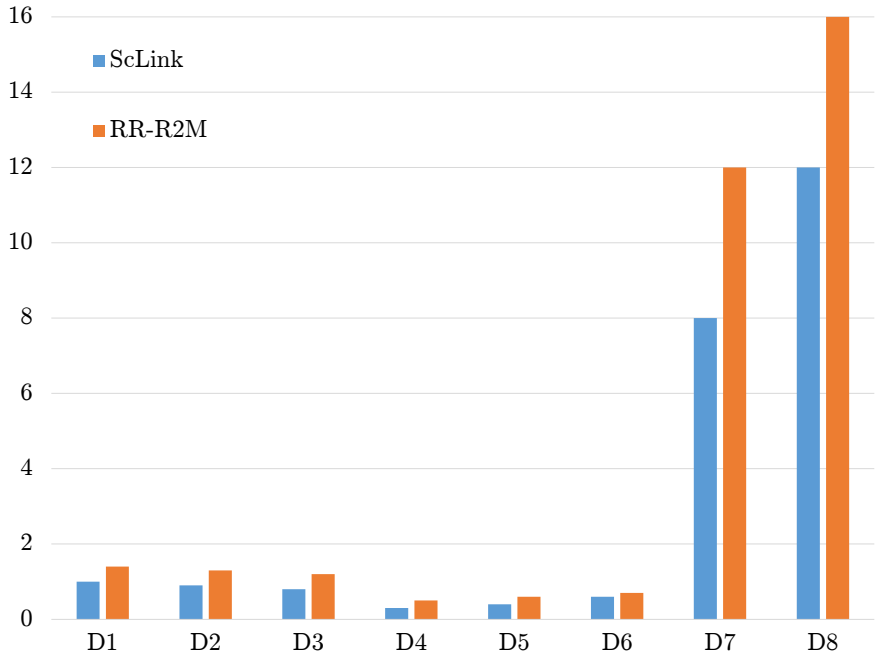


FIGURE 6.6: Literal similarity estimation time of *ScLink* and RR-*R2M* (in second).

10% training data is given, are reported in Figure 6.5 and 6.6. According to this figure, *ScLink* is much faster than RR-*R2M*, especially on large datasets. The time for executing *cLearn* is almost half of the training time of *R2M* and Random forest, together. The similarity estimation of *ScLink* is always lower than using classifier. It confirms the effect of similarity function reduction, which benefits from the interpretation of the matching specification. However, the classifier, especially when being support by *R2M*, is significantly better than *ScLink*. Therefore, a classifier with *R2M* is recommended for small and medium repositories while *ScLink* is more suitable for very large datasets.

6.6 Summary

We presented ranking feature *R2M*, whose objective is to enhance the classification-based instance matching system. We describe the design of *R2M* as well as optimization technique for training the model of *R2M* using curated data. We install *R2M* feature into original feature vectors to evaluate the performance of the classifier. We reported the detail analyses and experiments for evaluating *R2M*. The experimental results demonstrated the effectiveness of *R2M* and confirmed its importance in classification-based matching. By using *R2M*, the classifier significantly improves the F1 score compared to baseline models and also *ScLink* system.

Chapter 7

Conclusion

This chapter summarizes our contributions and discuss the future work.

7.1 Discussion

Matching different instances of the same real-world entity is an important problem in the current period of data explosion. In this dissertation, we present a series of solutions to instance matching that target on the issues of heterogeneity, ambiguity, and scalability. The solutions are designed for two different scenarios: supervised and non-supervised. They are reflected by the framework, method, or algorithms, and implemented in different systems. This section discusses the achievements of the dissertation.

- We develop the core of instance matching systems, *ScSLINT* framework. *ScSLINT* is designed for scalability, extensibility, and portability. The architecture of *ScSLINT* is based on the general workflow of similarity-based instance matching systems. It provides the interface of property alignment, blocking, similarity function and aggregation, and the determiner. *ScSLINT* is portable for different input data such as relation database and linked data repository. *ScSLINT* is compatible with classification-based and specification-based matching by adjusting the determiner. Furthermore, *ScSLINT* is also compatible with the supervised matching scenario by modifying the specification creator. We evaluated *ScSLINT* on real large datasets, including a huge dataset linking the whole of DBpedia and Freebase. *ScSLINT* is far better than existing frameworks in both terms of memory and time consumption. Concretely, the speed of *ScSLINT* is at least 10 times faster and *ScSLINT* is the first framework evaluated with the full dataset of OAEI 2011 instance matching challenge as well as a trillion scale dataset. We further propose two specification-based systems, *ASL* and *ScLink*, whose scalability is strongly supported by *ScSLINT*.

- In order to cover any repository with any domain and schema, we develop *ASL*, an automatic schema-independent instance matching system. *ASL* works independently with the schema because it aligns the properties using a simple but effective heuristic. *ASL* also adopts the principle of stable marriage problem for finding the stable coreferences among the pairwise instances. For evaluating *ASL*, we constructed a diversified dataset from the links between DBpedia and Freebase. 246 subsets of different schemas are acquired. The experiments on this dataset demonstrated the schema-independent capability of *ASL*. Compared to recent state-of-the-art systems, *ASL* significantly outperforms in terms of performance and also in processing time. Another finding from the experiments is the usefulness of blocking technique using only the first token. It is sufficient to use this technique for real dataset without losing a considerable amount of pair completeness, in comparison to weighting and ranking methods.

- Targeting on the ambiguity, we propose *Modified-BM25* similarity metric for string values. *Modified-BM25* combines the measurements on different aspects of the given strings. First, it leverages the advantage of Jaccard index on set similarity. Second, it applies state-of-the-art BM25 weighting scheme. Third, unlike other metrics, *Modified-BM25* takes the relative token's order into account in order to improve the disambiguation ability. The combination of above factors is not a usual linear aggregation rather than a probability conjunction. *Modified-BM25* is included in *ScLink* and is evaluated in the context of this system. The experiment results on many datasets including real and large repositories showed the drastic improvements when applying *Modified-BM25*, compared to using only other similarities, such as TF-IDF cosine and Levenshtein.

- In order to facilitate the scalable supervised instance matching, we propose *ScLink* system. The originality of *ScLink* includes three main points. First, while other supervised systems skip the learning of blocking scheme, *ScLink* follows a more reasonable workflow, in which the blocking scheme is optimized for a better reduction on the number of candidates. Second, *ScLink* is equipped with *minBlock* algorithm that learns the optimal blocking scheme. Third, *ScLink* uses *cLearn* for finding the specification that works best for the input repositories. *cLearn* utilizes a heuristic-based searching algorithm on the basis of apriori principle. The experiments on 15 datasets demonstrated the performance of *minBlock* and *cLearn* algorithms. *minBlock* helps generate a compact set of candidates with losing almost none of pair completeness. Meanwhile, *cLearn* finds the specification that significantly improves the F1 score against state-of-the-art algorithms. *ScLink* also consistently outperforms other systems in many comparisons.

- While other systems simply apply traditional classier for instance matching, the ranking of instance pairs with respective to each source instance is ignored. In order to benefit the generalization capability of machine learning classification and include ranking nature of instance matching, we propose *R2M* feature. *R2M* represents the ranking of instance pairs by probabilistic values. The model generating *R2M* is trained by optimization algorithms. *R2M* is evaluated with different state-of-the-art classifiers, including Linear regression, J48 decision tree, SVM, and Random forest. *R2M* demonstrated the outstanding ability of contribution to the generalization of all tested classifiers. The inclusion of *R2M* also significantly enhances the base classifiers. Moreover, the best classifier improved by *R2M* significantly outperforms *ScLink* and other systems.

- We developed different systems for various scenarios of instance matching. In summary, it is recommended to use *ASL* for non-supervised matching tasks, where the

schema-independence is required. The *ScLink* and classifier with *R2M* are suitable for supervised tasks. The accuracy of classifier with *R2M* is better than *ScLink* but *ScLink* is more scalable. Therefore, the application of *ScLink* and *R2M* is based on the consideration of computational cost and the expected accuracy.

- It is possible to combine specification-based and classification-based methods in the scenario of supervised matching (e.g., the combination of *ScLink* and *R2M*). Specification learning is responsible for finding the optimal similarity functions and the classifier learning can use the result of those functions as the input similarity vector. Such combination could reduce the complexity of the classifier. However, it is potentially redundant and conflict because both of the learning phases focus on the same objective. The training of classifiers already includes the selection of optimal similarity functions. Also, the usefulness of similarities is treated differently in accordance with each learning mechanism. In other words, the selection of the first learning phase can affect the performance of the second phase. Therefore, putting both learning phases together requires a study effort on the model combination.

7.2 Outlook

We are witnessing the fastest period of the development of digital data. This development definitely will be more accelerated over the time. Instance matching will soon have to cope with more difficult scalability issue. The architecture of *ScSLINT* and other specification-based systems allow parallel processing, at least for the most expensive step, the literal similarity estimation. The distributed computing technology is a powerful tool that can be used to scale-up the instance matching systems without touching their detailed technical aspects. For example, it is feasible to convert the current multi-threading settings installed in *ScSLINT* into the mechanism of high-performance parallelization frameworks.

Supervised instance matching has demonstrated the impressive performance. However, training data is a requirement for this approach. Although it is empirically evaluated that the amount of training data is not necessary to be large, the construction of high quality curated data does take a cost. Therefore, it is important to simplify the training data creation process. For example, active learning is a feasible candidate. In order to use 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. Active learning was used in some instance matching systems [61, 99] but this issue is

not satisfactorily resolved. Transfer learning is also a promising tool. 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 coreferences between similar repositories (e.g., on the domain, schema, and provider). Transfer learning is preliminarily evaluated that to work with instance matching problem [134]. However, the issue of identifying exactly the equivalent properties between the trained repositories and the unknown repositories having different schema remains unsolved. Furthermore, a study on measuring the similarity of two repositories is important for applying a model trained on a dataset to other datasets.

We presented the contributions in different approaches of supervised instance matching, for specification-based and classification-based matching. We have shown that classifier is doing better than heuristic search on many datasets. However, classification-based matching is more computational complicated and there are also cases that specification-based matching outperforms. We envision that their combination can deliver the best accuracy if the combined model is carefully optimized. Broadly, it is not only about combining *ScLink* and *R2M*-enabled classifiers, but also to select, to combine, and to tune the best fusion of many matchers, for all fashions of matching, including automatic, unsupervised, and supervised. In order to obtain that goal, the assessment on strength and weakness of each matcher, for each particular situation, has to be carried in the first place. Following that is the self-configuring or tuning the individual matcher and their combination.

7.3 Conclusion

Recently, linked data is proposed as a standard of future web-based data. The instances are organized in a structured manner that computers can share and understand easily. The linked data can help improving many applications such as reasoner, search engine, question and answering system, etc... Nevertheless, the decentralization of the Web is irreversible so that the entities will always be distributed and locally incomplete. Without the support of instance matching, it is impossible to draw the full picture of an entity. Furthermore, the introduction of linked data will not terminate the publication of unstructured text, the richest information so far. Linking the mentions in the text is also important to build up the extensive knowledge-base. Therefore, the instance matching will always be indispensable and worth to incisively study. Our proposed methods demonstrated the attractive effectiveness, but within the scope of this dissertation, not all problems of instance matching are resolved, such like those discussed previously. We envision that the maturity of instance matching will completely help to build a globally

interconnected data, not only for linked data, but also for all kinds of data including the unstructured text.

Bibliography

[1] R. Agrawal, R. Srikant, et al. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases*, volume 1215, pages 487–499, 1994.

[2] A. Arasu, V. Ganti, and R. Kaushik. Efficient exact set-similarity joins. In *Proceedings of the 32nd international conference on Very large data bases*, pages 918–929. VLDB Endowment, 2006.

[3] S. Araujo, A. De Vries, and D. Schwabe. SERIMI results for OAEI 2011. In *Proceedings of the 6th workshop on Ontology Matching*, pages 212–219, 2011.

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

[5] S. Araujo, D. T. Tran, A. DeVries, J. Hidders, and D. Schwabe. SERIMI: Class-based disambiguation for effective instance matching over heterogeneous web data. In *Proceedings of the 15th SIGMOD workshop on the Web and Databases*, pages 25–30, 2012.

[6] A. Bagga and B. Baldwin. Entity-based cross-document coreferencing using the vector space model. In *Proceedings of the 17th International Conference on Computational linguistics*, pages 79–85. Association for Computational Linguistics, 1998.

[7] S. Banerjee and T. Pedersen. Extended gloss overlaps as a measure of semantic relatedness. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, volume 3, pages 805–810, 2003.

[8] R. Baxter, P. Christen, and T. Churches. A comparison of fast blocking methods for record linkage. In *Proceedings of the SIGKDD workshop on Data Cleaning, Record Linkage and Object Consolidation*. ACM, 2003.

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

[10] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. E. Whang, and J. Widom. Swoosh: a generic approach to entity resolution. *The VLDB Journal—The International Journal on Very Large Data Bases*, 18(1):255–276, 2009.

[11] I. Bhattacharya and L. Getoor. Iterative record linkage for cleaning and integration. In *Proceedings of the 9th SIGMOD workshop on Research Numbers in Data Mining and Knowledge Discovery*, pages 11–18. ACM, 2004.

[12] I. Bhattacharya and L. Getoor. A latent dirichlet model for unsupervised entity resolution. In *Proceedings of the 6th SIAM International Conference on Data Mining*, pages 47–58. SIAM, 2006.

[13] I. Bhattacharya and L. Getoor. Collective entity resolution in relational data. *ACM Transactions on Knowledge Discovery from Data*, 1(1):1–36, 2007.

[14] M. Bilenko, B. Kamath, and R. J. Mooney. Adaptive blocking: Learning to scale up record linkage. In *Proceedings of the 6th International Conference on Data Mining*, pages 87–96, 2006.

[15] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 39–48. ACM, 2003.

[16] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pages 89–96. ACM, 2005.

[17] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pages 129–136. ACM, 2007.

[18] W. Chen, T.-Y. Liu, Y. Lan, Z.-M. Ma, and H. Li. Ranking measures and loss functions in learning to rank. In *Proceedings of Advances in Neural Information Processing Systems*, pages 315–323, 2009.

[19] N. Choi, I.-Y. Song, and H. Han. A survey on ontology mapping. *ACM Sigmod Record*, 35:34–41, 2006.

[20] P. Christen. 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*, pages 151–159. ACM, 2008.

[21] P. Christen. 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*, pages 511–518. Springer, 2008.

[22] P. Christen. 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*, volume 80, pages 17–25, 2008.

[23] P. Christen. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Transactions on Knowledge and Data Engineering*, 24(9):1537–1555, 2012.

[24] P. Christen and R. W. Gayler. Adaptive temporal entity resolution on dynamic databases. In *Proceedings of the 17th Pacific-Asia conference on Advances in Knowledge Discovery and Data Mining*, pages 558–569. Springer, 2013.

[25] R. L. Cilibrasi and P. Vitanyi. The google similarity distance. *IEEE Transactions on Knowledge and Data Engineering*, 19(3):370–383, 2007.

[26] W. S. Cooper, F. C. Gey, and D. P. Dabney. Probabilistic retrieval based on staged logistic regression. In *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 198–210. ACM, 1992.

[27] I. F. Cruz, F. P. Antonelli, and C. Stroe. AgreementMaker: Efficient matching for large real-world schemas and ontologies. In *Proceedings of the VLDB Endowment*, volume 2, pages 1586–1589, 2009.

[28] P. Cudré-Mauroux, P. Haghani, M. Jost, K. Aberer, and H. De Meer. idmesh: graph-based disambiguation of linked data. In *Proceedings of the 18th International Conference on World Wide Web*, pages 591–600. ACM, 2009.

[29] N. Dalvi, V. Rastogi, A. Dasgupta, A. Das Sarma, and T. Sarlós. Optimal hashing schemes for entity matching. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 295–306, 2013.

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

[31] L. R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945.

[32] X. Dong, A. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *Proceedings of the 24th SIGMOD International Conference on Management of Data*, pages 85–96. ACM, 2005.

[33] H. L. Dunn. Record linkage. *American Journal of Public Health and the Nations Health*, 36(12):1412–1416, 1946.

[34] M. Ehrig and Y. Sure. Ontology mapping–an integrated approach. In *The Semantic Web: Research and Applications*, pages 76–91. Springer, 2004.

[35] M. G. Elfeky, V. S. Verykios, and A. K. Elmagarmid. TAILOR: A record linkage toolbox. In *Proceedings of the 18th International Conference on Data Engineering*, pages 17–28. IEEE, 2002.

[36] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Transaction on Knowledge and Data Engineering*, 19:1–16, 2007.

[37] J. Euzenat, A. Ferrara, W. R. van Hague, L. Hollink, C. Meilicke, A. Nikolov, F. Scharffe, P. Shvaiko, H. Stuckenschmidt, O. Sváb-Zamazal, and C. T. dos Santos. Final results of the ontology alignment evaluation initiative 2011. In *Proceedings of the 6th workshop on Ontology Matching*, pages 85–113, 2011.

[38] D. Faria, C. Pesquita, E. Santos, M. Palmonari, I. F. Cruz, and F. M. Couto. The agreementmakerlight ontology matching system. In *On the Move to Meaningful Internet Systems: OTM 2013 Conferences*, pages 527–541. Springer, 2013.

[39] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.

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

[41] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *The Journal of machine learning research*, 4:933–969, 2003.

[42] N. Fuhr. Optimum polynomial retrieval functions based on the probability ranking principle. *ACM Transactions on Information Systems (TOIS)*, 7:183–204, 1989.

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

[44] L. Getoor and C. P. Diehl. Link mining: a survey. *ACM SIGKDD Explorations Newsletter*, 7(2):3–12, 2005.

[45] C. H. Gooi and J. Allan. Cross-document coreference on a large scale corpus. Technical report, DTIC Document, 2004.

[46] A. Halevy, A. Rajaraman, and J. Ordille. Data integration: the teenage years. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, pages 9–16. VLDB Endowment, 2006.

[47] R. Hall, C. Sutton, and A. McCallum. Unsupervised deduplication using cross-field dependencies. In *Proceedings of the 14th SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 310–317. ACM, 2008.

[48] O. Hassanzadeh and M. Consens. Linked movie database. In *Proceedings of WWW' 09 2nd Workshop on Linked Data on the Web*, 2009.

[49] Hassell, Joseph and Aleman-Meza, Boanerges and Arpinar, I Budak. *Ontology-driven automatic entity disambiguation in unstructured text*. Springer, 2006.

[50] M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. *ACM SIGMOD Record*, 24:127–138, 1995.

[51] M. A. Hernández and S. J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data mining and knowledge discovery*, 2:9–37, 1998.

[52] G. Hirst and D. St-Onge. Lexical chains as representations of context for the detection and correction of malapropisms. *The MIT Press*, 305:305–332, 1998.

[53] J. Hoffart, M. A. Yosef, I. Bordino, H. Fürstenau, M. Pinkal, M. Spaniol, B. Taneva, S. Thater, and G. Weikum. Robust disambiguation of named entities in text. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 782–792. Association for Computational Linguistics, 2011.

[54] A. Hogan, A. Zimmermann, J. Umbrich, A. Polleres, and S. Decker. 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, 2012.

[55] W. Hu, J. Chen, G. Cheng, and Y. Qu. Objectcoref & falcon-ao: results for oaei 2010. In *Proceedings of the 5th workshop on Ontology Matching*, pages 158–165, 2010.

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

[57] W. Hu and Y. Qu. Falcon-ao: A practical ontology matching system. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6:237–239, 2008.

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

[59] E. Ioannou, O. Papapetrou, D. Skoutas, and W. Nejdl. Efficient semantic-aware detection of near duplicate resources. In *The Semantic Web: Research and Applications*, pages 136–150. Springer, 2010.

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

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

[62] R. Isele, A. Jentzsch, and C. Bizer. Efficient multidimensional blocking for link discovery without losing recall. In *Proceedings of the 14th SIGMOD workshop on the Web and Databases*, 2011.

[63] A. Islam and D. Inkpen. Semantic text similarity using corpus-based word similarity and string similarity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2(2):10, 2008.

[64] P. Jaccard. *Étude comparative de la distribution florale dans une portion des Alpes et des Jura*, volume 37. Impr. Corbaz, 1901.

[65] M. A. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, 84:414–420, 1989.

[66] Y. R. Jean-Mary, E. P. Shironoshita, and M. R. Kabuka. Ontology matching with semantic verification. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7:235–251, 2009.

[67] J. J. Jiang and D. W. Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. In *Proceedings of 10th International Conference on Research in Computational Linguistics, ROCLING' 97*, pages 19–33, 1997.

[68] E. Jiménez-Ruiz and B. C. Grau. Logmap: Logic-based and scalable ontology matching. In *The Semantic Web – ISWC 2011*, pages 273–288. Springer, 2011.

[69] E. Jiménez-Ruiz, B. C. Grau, Y. Zhou, and I. Horrocks. Large-scale interactive ontology matching: Algorithms and implementation. In *Proceedings of the European Conference on Artificial Intelligence*, volume 242, pages 444–449, 2012.

[70] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142. ACM, 2002.

[71] Y. Kalfoglou and M. Schorlemmer. Ontology mapping: the state of the art. *The Knowledge Engineering Review*, 18:1–31, 2003.

[72] M. Kejriwal and D. P. Miranker. An unsupervised algorithm for learning blocking schemes. In *Proceedings of the 13th International Conference on Data Mining*, pages 340–349. IEEE, 2013.

[73] M. Kejriwal and D. P. Miranker. A two-step blocking scheme learner for scalable link discovery. In *Proceedings of the 9th workshop on Ontology Matching*, 2014.

[74] M. Kejriwal and D. P. Miranker. Semi-supervised instance matching using boosted classifiers. In *Proceedings of the 12th Extended Semantic Web Conference*, volume 9088 of *LNCS*, pages 388–402. Springer, 2015.

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

[76] H. Köpcke, A. Thor, and E. Rahm. Evaluation of entity resolution approaches on real-world match problems. In *Proceedings of the VLDB Endowment*, volume 3, pages 484–493. VLDB Endowment, 2010.

[77] N. Koudas, S. Sarawagi, and D. Srivastava. Record linkage: similarity measures and algorithms. In *Proceedings of the 25th SIGMOD International Conference on Management of Data*, pages 802–803. ACM, 2006.

[78] C. Leacock, G. A. Miller, and M. Chodorow. Using corpus statistics and wordnet relations for sense identification. *Computational Linguistics*, 24:147–165, 1998.

[79] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Proceedings of Soviet Physics Doklady*, volume 10, pages 707–710, 1966.

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

[81] W.-S. Li and C. Clifton. SEMINT: a tool for identifying attribute correspondences in heterogeneous databases using neural networks. *Data Knowledge and Engineering*, 33:49–84, 2000.

[82] D. Lin. An information-theoretic definition of similarity. In *Proceedings of the 15th International Conference on Machine Learning*, volume 98, pages 296–304, 1998.

[83] T.-Y. Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3:225–331, 2009.

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

[85] J. Lu, C. Lin, W. Wang, C. Li, and H. Wang. String similarity measures and joins with synonyms. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 373–384. ACM, 2013.

[86] K. Lyko, K. Höffner, R. Speck, A.-C. N. Ngomo, and J. Lehmann. SAIM–One step closer to zero-configuration link discovery. In *The Semantic Web: ESWC 2013 Satellite Events*, LNCS, pages 167–172. Springer, 2013.

[87] B. Marshall, H. Chen, and T. Madhusudan. Matching knowledge elements in concept maps using a similarity flooding algorithm. *Decision Support Systems*, 42(3):1290–1306, 2006.

[88] A. McCallum, K. Nigam, and L. H. Ungar. 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*, pages 169–178. ACM, 2000.

[89] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings of 18th International Conference on Data Engineering*, pages 117–128. IEEE, 2002.

[90] P. N. Mendes, M. Jakob, A. García-Silva, and C. Bizer. Dbpedia spotlight: shedding light on the web of documents. In *Proceedings of the 7th International Conference on Semantic Systems*, pages 1–8. ACM, 2011.

[91] L. Meng, R. Huang, and J. Gu. A review of semantic similarity measures in wordnet. *International Journal of Hybrid Information Technology*, 6:1–12, 2013.

[92] M. Michelson and C. A. Knoblock. Learning blocking schemes for record linkage. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 440. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

[93] R. Mihalcea, C. Corley, and C. Strapparava. Corpus-based and knowledge-based measures of text semantic similarity. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 6, pages 775–780, 2006.

[94] G. A. Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38:39–41, 1995.

[95] E. Moreau, F. Yvon, and O. Cappé. Robust similarity measures for named entities matching. In *Proceedings of the 22nd International Conference on Computational Linguistics*, volume 1, pages 593–600. Association for Computational Linguistics, 2008.

[96] H. B. Newcombe, J. Kennedy, S. Axford, and A. P. James. Automatic linkage of vital records. *Science*, 130(3381):954–959, 1959.

[97] A.-C. N. Ngomo. Learning conformation rules for linked data integration. In *Proceedings of the 7th workshop on Ontology Matching*, pages 13–24, 2012.

[98] A.-C. N. Ngomo and S. Auer. 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*, pages 2312–2317, 2011.

[99] A. C. N. Ngomo, J. Lehmann, S. Auer, and K. Höffner. RAVEN - Active learning of link specifications. In *Proceedings of the 6th workshop on Ontology Matching*, pages 25–36, 2011.

[100] A.-C. N. Ngomo and K. Lyko. EAGLE: Efficient active learning of link specifications using genetic programming. In *Proceedings of the 9th Extended Semantic Web Conference*, volume 7295 of *LNCS*, pages 149–163. Springer, 2012.

[101] A.-C. N. Ngomo and K. Lyko. Unsupervised learning of link specifications: Deterministic vs. non-deterministic. In *Proceedings of the 8th workshop on Ontology Matching*, pages 25–36, 2013.

[102] K. Nguyen and R. Ichise. SLINT+ results for OAEI 2013 instance matching. In *Proceedings of the 8th workshop on Ontology Matching*, pages 177–183, 2013.

[103] K. Nguyen and R. Ichise. An effective configuration learning algorithm for entity resolution. In *Proceedings of the 10th workshop on Ontology Matching*, pages 177–183, 2015.

[104] K. Nguyen and R. Ichise. An effective configuration learning algorithm for entity resolution. In *Proceedings of the 10th workshop on Ontology Matching*, 2015.

[105] K. Nguyen and R. Ichise. A heuristic approach for configuration learning of supervised instance matching. In *Proceedings of 14th International Semantic Web Conference Posters and Demonstrations Track*, 2015.

[106] K. Nguyen and R. Ichise. Heuristic-based configuration learning for linked data instance matching. In *Proceedings of the 5th Joint International Semantic Technology Conference*, 2015.

[107] K. Nguyen and R. Ichise. ScSLINT: Time and memory efficient interlinking framework for linked data. In *Proceedings of the 14th International Semantic Web Conference Posters and Demonstrations Track*, 2015.

[108] K. Nguyen and R. Ichise. Automatic schema-independent linked data instance matching system. *International Journal of Semantic Web and Information System*, 2016.

[109] K. Nguyen and R. Ichise. Linked data entity resolution system enhanced by configuration learning algorithm. *IEICE Transactions on Information and Systems*, E99-D, 2016.

[110] K. Nguyen and R. Ichise. Ranking feature for classifier-based instance matching. In *Proceedings of 15th International Semantic Web Conference Posters and Demonstrations Track*, 2016.

[111] K. Nguyen and R. Ichise. Sclink: supervised instance matching system for heterogeneous repositories. *Intelligent Information Systems*, 2016.

[112] K. Nguyen, R. Ichise, and B. Le. Interlinking linked data sources using a domain-independent system. In *Proceedings of the 2nd Joint International Semantic Technology*, volume 7774 of *LNCS*, pages 113–128. Springer, 2012.

[113] K. Nguyen, R. Ichise, and B. Le. Slint: a schema-independent linked data interlinking system. In *Proceedings of 7th International Workshop on Ontology Matching*, pages 1–12, 2012.

[114] K. Nguyen, R. Ichise, and H.-B. Le. Learning approach for domain-independent linked data instance matching. In *Proceedings of the SIGKDD 2nd workshop on Mining Data Semantics*, pages 7–15. ACM, 2012.

[115] M. Niepert, C. Meilicke, and H. Stuckenschmidt. A probabilistic-logical framework for ontology matching. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, pages 1413–1418, 2010.

[116] A. Nikolov, M. d'Aquin, and E. Motta. Unsupervised learning of link discovery configuration. In *Proceedings of the 9th Extended Semantic Web Conference*, volume 7295 of *LNCS*, pages 119–133. Springer, 2012.

[117] X. Niu, S. Rong, Y. Zhang, and H. Wang. Zhishi.links results for OAEI 2011. In *Proceedings of the 6th workshop on Ontology Matching*, pages 220–227, 2011.

[118] X. Niu, X. Sun, H. Wang, S. Rong, G. Qi, and Y. Yu. Zhishi. me-weaving chinese linking open data. In *The Semantic Web – ISWC 2011*, pages 205–220. Springer, 2011.

[119] J. Noessner, M. Niepert, C. Meilicke, and H. Stuckenschmidt. Leveraging terminological structure for object reconciliation. In *The Semantic Web: Research and Applications*, pages 334–348. Springer, 2010.

[120] G. Papadakis, E. Ioannou, C. Niederée, and P. Fankhauser. Efficient entity resolution for large heterogeneous information spaces. In *Proceedings of the 4th International Conference on Web Search and Data Mining*, pages 535–544. ACM, 2011.

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

[122] G. Papadakis, G. Papastefanatos, and G. Koutrika. Supervised meta-blocking. In *Proceedings of the VLDB Endowment*, volume 7, pages 1929–1940. VLDB Endowment, 2014.

[123] S. Patwardhan. *Incorporating dictionary and corpus information into a context vector measure of semantic relatedness.* PhD thesis, University of Minnesota, Duluth, 2003.

[124] T. Pedersen, S. Patwardhan, and J. Michelizzi. Wordnet:: Similarity: measuring the relatedness of concepts. In *Proceedings of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies Demonstration*, pages 38–41. Association for Computational Linguistics, 2004.

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

[126] J. C. Pinheiro and D. X. Sun. Methods for linking and mining massive heterogeneous databases. In *Proceedings of the 4th SIGKDD Conference on Knowledge Discovery and Data Mining*, 1998.

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

[128] T. Qin, T.-Y. Liu, J. Xu, and H. Li. Letor: A benchmark collection for research on learning to rank for information retrieval. *Information Retrieval*, 13:346–374, 2010.

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

[130] Y. Raimond, C. Sutton, and M. Sandler. Automatic interlinking of music datasets on the semantic web. In *Proceedings of the workshop on Linked Data On the Web*, Beijing, China, 2008.

[131] P. Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 448–453, 1995.

[132] S. E. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. In *Proceedings of the 3rd Text Retrieval Conference*, pages 109–123, 1994.

[133] M. A. Rodríguez and M. J. Egenhofer. Determining semantic similarity among entity classes from different ontologies. *IEEE Transactions on Knowledge and Data Engineering*, 15(2):442–456, 2003.

[134] S. Rong, X. Niu, W. E. Xiang, H. Wang, Q. Yang, and Y. Yu. A machine learning approach for instance matching based on similarity metrics. In *Proceedings of the 11th International Semantic Web Conference*, volume 7649 of *LNCS*, pages 460–475. Springer, 2012.

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

[136] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24:513–523, 1988.

[137] C. Sarasua, E. Simperl, and N. F. Noy. Crowdmap: Crowdsourcing ontology alignment with microtasks. In *The Semantic Web – ISWC 2012*, pages 525–541. Springer, 2012.

[138] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *Proceedings of the 8th SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 269–278, New York, USA, 2002. ACM.

[139] F. Scharffe, Y. Liu, and C. Zhou. RDF-AI: An architecture for RDF datasets matching, fusion and interlink. In *Proceedings of the workshop on Identity, Reference, and Knowledge Representation*, Pasadena, USA, 2009. AAAI Press.

[140] M. Schmachtenberg, C. Bizer, and H. Paulheim. Adoption of the linked data best practices in different topical domains. In *Proceedings of the 13th International Semantic Web Conference*, volume 8796 of *LNCS*, pages 245–260. Springer, 2014.

[141] A. Schultz, A. Matteini, R. Isele, C. Bizer, and C. Becker. Ldif - linked data integration framework. In *Proceeding of ISWC' 11 2nd Workshop on Consuming Linked Data*, Bonn, Germany, 2011. CEUR-WS.org.

[142] D. Sculley. Combined regression and ranking. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 979–988. ACM, 2010.

[143] T. Sheila, C. Knoblock, and S. Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *Proceedings of the 8th SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 350–359. ACM, 2002.

[144] P. Shvaiko and J. Euzenat. Ontology matching: state of the art and future challenges. *IEEE Transactions on Knowledge and Data Engineering*, 25:158–176, 2013.

[145] G. G. Simpson. Notes on the measurement of faunal resemblance. *American Journal of Science*, 258:300–311, 1960.

[146] J. M. Smith, P. A. Bernstein, U. Dayal, N. Goodman, T. Landers, K. W. T. Lin, and E. Wong. Multibase: Integrating heterogeneous distributed database systems. In *Proceedings of the 9th National Computer Conference*, AFIPS '81, pages 487–499. ACM, 1981.

[147] D. Song and J. Heflin. Domain-independent entity coreference in RDF graphs. In *Proceedings of the 19th International Conference on Information and Knowledge Management* , pages 1821–1824, Toronto, Canada, 2010. ACM.

[148] D. Song and J. Heflin. Automatically generating data linkages using a domain-independent candidate selection approach. In *Proceedings of the 10th International Semantic Web Conference*, volume 7031 of *LNCS*, pages 649–664. Springer, 2011.

[149] T. Sørensen. A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on danish commons. *Kongelige Danske Videnskabernes Selskab*, 5:1–34, 1948.

[150] T. Soru and A.-C. N. Ngomo. A comparison of supervised learning classifiers for link discovery. In *Proceedings of the 10th International Conference on Semantic Systems*, pages 41–44. ACM, 2014.

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

[152] A. Thor and E. Rahm. MOMA-a mapping-based object matching system. In *Proceedings of the 3rd Biennial Conference on Innovative Data Systems Research*, pages 247–258, 2007.

[153] J. Urbani, S. Kotoulas, J. Maassen, F. Van Harmelen, and H. Bal. OWL reasoning with webpie: calculating the closure of 100 billion triples. In *Proceedings of the 7th European Semantic Web Conference*, volume 5554 of *LNCS*, pages 213–227. Springer, 2010.

[154] N. Vesdapunt, K. Bellare, and N. Dalvi. Crowdsourcing algorithms for entity resolution. In *Proceedings of the VLDB Endowment*, volume 7, pages 1071–1082. VLDB Endowment, 2014.

[155] J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov. Discovering and maintaining links on the web of data. In *Proceedings of the 8th International Semantic Web Conference*, volume 5823 of *LNCS*, pages 650–665. Springer, 2009.

[156] J. Wang, J. Feng, and G. Li. Trie-join: Efficient trie-based string similarity joins with edit-distance constraints. *Proceedings of the VLDB Endowment*, 3:1219–1230, 2010.

[157] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. CrowdER: Crowdsourcing entity resolution. In *Proceedings of the VLDB Endowment*, volume 5, pages 1483–1494. VLDB Endowment, 2012.

[158] J. Wang, G. Li, and J. Fe. Fast-join: An efficient method for fuzzy token matching based string similarity join. In *Proceedings fo IEEE 27th International Conference on Data Engineering (ICDE)*, pages 458–469. IEEE, 2011.

[159] S. E. Whang, D. Menestrina, G. Koutrika, M. Theobald, and H. Garcia-Molina. Entity resolution with iterative blocking. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 219–232. ACM, 2009.

[160] W. Winkler. String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage. In *Proceedings of the American Statistical Association Survey Research Methods Section*, pages 354–359, 1990.

[161] W. E. Winkler. Approximate string comparator search strategies for very large administrative lists. *Statistics*, 2, 2005.

[162] Z. Wu and M. Palmer. Verbs semantics and lexical selection. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, pages 133–138. Association for Computational Linguistics, 1994.

[163] F. Xia, T.-Y. Liu, J. Wang, W. Zhang, and H. Li. Listwise approach to learning to rank: theory and algorithm. In *Proceedings of the 25th international conference on Machine learning*, pages 1192–1199. ACM, 2008.

[164] C. Xiao, W. Wang, and X. Lin. Ed-Join: an efficient algorithm for similarity joins with edit distance constraints. *Proceedings of the VLDB Endowment*, 1:933–944, 2008.

[165] C. Xiao, W. Wang, X. Lin, J. X. Yu, and G. Wang. Efficient similarity joins for near-duplicate detection. *ACM Transactions on Database Systems*, 36(3):15:1–15:41, 2011.

[166] S. Yan, D. Lee, M.-Y. Kan, and L. C. Giles. Adaptive sorted neighborhood methods for efficient record linkage. In *Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries*, pages 185–194. ACM, 2007.

Appendix A

The repositories constructed from DBpedia

No.	#class	#instance	#property	#fact
1	Convention	554	105	53197
2	Train	1087	195	73926
3	Game	1276	253	75338
4	ProgrammingLanguage	499	150	44498
5	Locomotive	2781	267	202039
6	InformationAppliance	931	275	80775
7	Comics	1982	368	153699
8	Galaxy	578	81	28482
9	Ligament	195	33	6203
10	Vein	232	39	9040
11	Muscle	281	56	15675
12	Nerve	338	40	14715
13	RadioProgram	933	252	76486
14	Diocese	2372	238	181268
15	Artery	370	44	16217
16	Musical	1198	158	100225
17	Archaea	171	50	8698
18	Cycad	173	65	9468
19	LunarCrater	1475	32	55119
20	SpaceMission	441	283	44963

21	SupremeCourtOfTheUnitedStatesCase	2479	140	150765
22	Food	3397	260	187106
23	Brain	523	60	25876
24	Race	515	70	55018
25	FilmFestival	536	91	46944
26	Star	2265	253	143996
27	Holiday	607	188	51097
28	CyclingTeam	222	59	28662
29	Grape	349	60	23618
30	Bacteria	352	95	21713
31	GreenAlga	356	55	17873
32	Moss	384	68	19435
33	WrestlingEvent	878	88	103485
34	MusicFestival	909	158	128598
35	BasketballLeague	306	126	33836
36	ChemicalCompound	8404	258	367639
37	GrandPrix	1154	128	159306
38	Website	2870	398	170292
39	Legislature	1011	292	119549
40	Racecourse	162	47	11556
41	Software	9906	616	657353
42	Conifer	720	103	52097
43	Fern	833	104	55322
44	Artwork	3412	216	171147
45	Aircraft	8946	816	764742
46	FootballMatch	2064	553	224005
47	MotorsportSeason	2069	153	196644
48	AnimangaCharacter	342	96	27900
49	AnatomicalStructure	2229	112	105651
50	Canal	269	152	22368
51	TelevisionStation	7093	604	741672
52	BasketballTeam	857	260	87012
53	Tunnel	298	177	21387
54	Automobile	7796	450	527705
55	SportsLeague	955	290	85950
56	HockeyTeam	962	220	79980
57	LawFirm	446	107	32884
58	NationalFootballLeagueSeason	3003	322	324975
59	BeachVolleyballPlayer	127	65	9243

60	WineRegion	334	129	32523
61	OlympicEvent	3070	303	283063
62	Planet	11823	227	674019
63	Cave	345	117	15833
64	SportsTeam	1092	228	65317
65	CultivatedVariety	1474	155	49319
66	MountainRange	375	248	27180
67	TennisTournament	3864	143	207466
68	Museum	3634	515	266385
69	SoccerLeague	1341	357	131890
70	Restaurant	623	170	35990
71	Award	2337	184	301817
72	Bodybuilder	205	101	13604
73	FootballLeagueSeason	4895	644	746938
74	FloweringPlant	2247	225	142051
75	Drug	5368	245	347698
76	HorseTrainer	220	60	30537
77	SkiArea	575	404	37653
78	Skater	222	82	16819
79	SiteOfSpecialScientificInterestc	578	146	28689
80	Crustacean	2374	187	155189
81	Disease	5604	262	349112
82	RugbyClub	1848	394	159749
83	Device	848	247	65016
84	NetballPlayer	243	83	14620
85	Ship	25555	806	2149224
86	Place	636	871	56307
87	Venue	642	174	49028
88	SnookerPlayer	264	85	20545
89	Publisher	940	161	66612
90	SquashPlayer	267	82	16501
91	RadioStation	17997	528	1295033
92	Library	708	217	49547
93	Election	5586	478	740527
94	Medician	282	150	25139
95	RaceHorse	2988	122	337247
96	Prison	740	127	43080
97	PlayboyPlaymate	294	128	33034
98	Volcano	792	282	55445

99	RadioHost	314	124	26477
100	Castle	816	114	40910
101	LacrossePlayer	339	473	28496
102	TradeUnion	1489	106	77258
103	Species	3838	266	194475
104	Theatre	981	242	82117
105	FashionDesigner	379	171	30923
106	Hotel	1045	261	68432
107	Chef	405	139	29924
108	Jockey	410	62	60357
109	AmateurBoxer	413	71	30038
110	WorldHeritageSite	1102	701	86411
111	DartsPlayer	436	124	35143
112	SumoWrestler	441	112	35124
113	TableTennisPlayer	445	104	24348
114	Ambassador	450	218	45584
115	BadmintonPlayer	465	102	33961
116	Curler	473	85	34311
117	MilitaryConflict	11431	599	1191959
118	Cardinal	507	78	38671
119	Reptile	5408	241	354575
120	NationalCollegiateAthleticAssociationAthlete	538	207	47704
121	Murderer	553	176	55585
122	PublicTransitSystem	1437	436	126397
123	Lighthouse	1471	411	87683
124	PokerPlayer	624	133	39792
125	BroadcastNetwork	1164	326	100406
126	Astronaut	628	173	78409
127	SoapCharacter	2408	316	242688
128	SpeedwayRider	639	94	55170
129	Amphibian	6894	161	367791
130	RecordLabel	2391	268	192142
131	Park	1756	300	100617
132	Economist	698	168	73962
133	Religious	708	256	63907
134	Engineer	731	130	62262
135	VideoGame	18459	1312	1564634
136	HollywoodCartoon	1486	125	104938
137	HandballPlayer	770	91	54713

138	GovernmentAgency	3283	454	237904
139	Fungus	8739	244	455119
140	NascarDriver	836	264	101428
141	Non-ProfitOrganisation	3494	551	237067
142	FormulaOneRacer	852	206	154137
143	ShoppingMall	2244	350	145723
144	VolleyballPlayer	868	108	63363
145	Airline	3124	281	245712
146	Hospital	2328	351	139507
147	RailwayLine	2385	360	223390
148	Senator	934	316	140388
149	PoliticalParty	3953	503	328632
150	FictionalCharacter	3635	638	326379
151	PowerStation	2481	353	162262
152	ReligiousBuilding	2609	378	177466
153	Comedian	1018	268	129230
154	ChessPlayer	1055	145	85031
155	Gymnast	1084	159	74033
156	Country	2833	1154	532156
157	ComicsCharacter	4531	341	391514
158	Bird	12520	254	853806
159	Bridge	3259	532	238029
160	Skier	1261	139	93045
161	MilitaryStructure	3320	642	251407
162	Architect	1291	183	124422
163	Philosopher	1300	291	188953
164	RacingDriver	1351	197	157481
165	Model	1354	320	102277
166	Island	3586	932	251204
167	Judge	1557	284	136820
168	AdultActor	1597	240	118760
169	RailwayStation	4315	115	275533
170	Organisation	7332	985	543795
171	BeautyQueen	1852	290	127152
172	HistoricBuilding	5072	664	393167
173	Anime	3796	266	148922
174	President	2196	538	265480
175	SoccerClub	17551	1265	1983516
176	ComicsCreator	2415	238	229314

177	Monarch	2420	411	218190
178	Swimmer	2471	151	191808
179	WrittenWork	1176	101	64019
180	FigureSkater	2513	217	222355
181	Mollusca	27275	205	1436131
182	GolfPlayer	2610	152	231548
183	Wrestler	2664	338	348314
184	Boxer	2698	289	269821
185	MartialArtist	2724	400	296391
186	Saint	2797	332	260589
187	Congressman	2810	385	320862
188	Play	1361	164	93542
189	ProtectedArea	7705	1027	547525
190	Stadium	7856	583	537669
191	GaelicGamesPlayer	3052	249	237546
192	MilitaryUnit	13934	930	1257670
193	TennisPlayer	3364	338	415706
194	Noble	3666	272	301864
195	Lake	9793	790	525491
196	Cyclist	3828	160	286396
197	HistoricPlace	9956	996	542293
198	Plant	40353	485	2344111
199	EducationalInstitution	16217	2449	1450649
200	EthnicGroup	3987	393	426654
201	TelevisionEpisode	7675	319	629111
202	ChristianBishop	4677	416	447796
203	Airport	12231	802	1001289
204	Mountain	12537	776	679720
205	CollegeCoach	5637	384	633710
206	AustralianRulesFootballPlayer	5840	235	349052
207	BasketballPlayer	6487	450	794076
208	MusicGenre	747	148	109021
209	School	28228	3272	2273072
210	Road	17869	578	1608072
211	GridironFootballPlayer	7092	466	541948
212	BritishRoyalty	7168	634	888858
213	Station	18961	777	1242293
214	City	20228	2084	2851897
215	Magazine	3767	457	255444

216	PopulatedPlace	22605	2382	1763185
217	River	24962	809	1284240
218	Animal	108008	743	5509533
219	Newspaper	5142	563	283100
220	AcademicJournal	5145	200	322986
221	RugbyPlayer	11098	805	936013
222	IceHockeyPlayer	11535	333	1222852
223	Cricketer	11614	531	1214698
224	AmericanFootballPlayer	11884	602	1228538
225	Company	42489	2292	3156275
226	SoccerManager	13363	366	1816804
227	TelevisionShow	25628	1981	2687463
228	Scientist	14612	687	1398494
229	Building	41694	1772	2531623
230	Athlete	17199	671	1109100
231	Politician	18784	1068	1778158
232	BaseballPlayer	19807	529	2154767
233	MilitaryPerson	23499	802	2568998
234	Artist	24892	1122	2385456
235	Song	4165	260	318650
236	MusicalArtist	37935	1539	4341409
237	OfficeHolder	38314	1854	3938759
238	Film	77768	1624	6060213
239	Village	119859	1129	7133643
240	Book	28127	819	1890537
241	SoccerPlayer	89078	734	9248270
242	Settlement	255565	3828	19236695
243	Band	28681	1063	2539429
244	Single	41763	555	3265027
245	Person	602293	4144	32335873
246	Album	116368	1133	8611068

Appendix B

The repositories constructed from Freebase

No.	#class	#instance	#property	#fact
1	conferences	1804	1414	234924
2	rail	4055	789	246981
3	games	3774	882	309461
4	computer	18998	2415	1833780
5	comic_books	9472	1267	1057210
6	astronomy	32590	1036	4161049
7	medicine	118438	3262	11123716
8	broadcast	43781	3127	8282088
9	religion	18028	1692	2303413
10	theater	36781	2073	3460678
11	biology	276338	2516	18396775
12	projects	110119	3886	6605095
13	law	21125	3344	2783895
14	food	16599	2959	5201620
15	time	122069	2489	7980299
16	sports	372776	4590	29880047
17	chemistry	16139	753	3431956
18	internet	49775	4889	10318436
19	government	143160	3731	12701720
20	location	1116911	4725	84126974

21	visual_art	68682	4077	4863237
22	aviation	26562	1445	2374442
23	fictional_universe	758989	6148	19822647
24	automotive	43213	974	1616546
25	business	817061	9883	34050312
26	people	2897510	6343	130081223
27	architecture	130970	3914	7673984
28	award	235979	6899	33420503
29	boats	27683	1412	1368877
30	organization	695810	7137	32252820
31	tv	1559059	5811	50204226
32	cvg	117919	3229	5262726
33	film	1470618	7946	69582765
34	book	6183219	2439785	158262579
35	music	25625291	4357	408994668

Appendix C

DF246 dataset

ID	Source	Target	Actual corerences
1	Convention	conferences	290
2	Train	rail	262
3	Game	games	816
4	ProgrammingLanguage	computer	378
5	Locomotive	rail	1477
6	InformationAppliance	computer	268
7	Comics	comic_books	1616
8	Galaxy	astronomy	510
9	Ligament	medicine	142
10	Vein	medicine	148
11	Muscle	medicine	255
12	Nerve	medicine	236
13	RadioProgram	broadcast	580
14	Diocese	religion	2144
15	Artery	medicine	300
16	Musical	theater	972
17	Archaea	biology	163
18	Cycad	biology	170
19	LunarCrater	astronomy	1475
20	SpaceMission	projects	385
21	SupremeCourtOfTheUnitedStatesCase	law	2239
22	Food	food	1634

23	Brain	medicine	301
24	Race	time	217
25	FilmFestival	time	439
26	Star	astronomy	1949
27	Holiday	time	285
28	CyclingTeam	sports	136
29	Grape	biology	156
30	Bacteria	biology	195
31	GreenAlga	biology	348
32	Moss	biology	350
33	WrestlingEvent	time	284
34	MusicFestival	time	444
35	BasketballLeague	sports	128
36	ChemicalCompound	chemistry	7538
37	GrandPrix	time	995
38	Website	internet	1968
39	Legislature	government	613
40	Racecourse	location	151
41	Software	computer	4136
42	Conifer	biology	701
43	Fern	biology	660
44	Artwork	visual_art	2609
45	Aircraft	aviation	4952
46	FootballMatch	time	786
47	MotorsportSeason	time	671
48	AnimangaCharacter	fictional_universe	349
49	AnatomicalStructure	medicine	1281
50	Canal	location	241
51	TelevisionStation	broadcast	3204
52	BasketballTeam	sports	418
53	Tunnel	location	159
54	Automobile	automotive	3015
55	SportsLeague	sports	382
56	HockeyTeam	sports	471
57	LawFirm	business	318
58	NationalFootballLeagueSeason	time	1264
59	BeachVolleyballPlayer	people	126
60	WineRegion	location	247
61	OlympicEvent	time	2991

62	Planet	astronomy	11732
63	Cave	location	177
64	SportsTeam	sports	464
65	CultivatedVariety	biology	432
66	MountainRange	location	330
67	TennisTournament	time	1673
68	Museum	architecture	2567
69	SoccerLeague	sports	542
70	Restaurant	business	213
71	Award	award	1427
72	Bodybuilder	people	199
73	FootballLeagueSeason	time	1523
74	FloweringPlant	biology	1965
75	Drug	medicine	3006
76	HorseTrainer	people	213
77	SkiArea	location	442
78	Skater	people	212
79	SiteOfSpecialScientificInterest	location	410
80	Crustacean	biology	1891
81	Disease	medicine	5434
82	RugbyClub	sports	1106
83	Device	business	688
84	NetballPlayer	people	242
85	Ship	boats	23635
86	Place	location	399
87	Venue	location	442
88	SnookerPlayer	people	264
89	Publisher	business	693
90	SquashPlayer	people	264
91	RadioStation	broadcast	14256
92	Library	location	224
93	Election	government	3417
94	Medician	people	277
95	RaceHorse	biology	2468
96	Prison	location	574
97	PlayboyPlaymate	people	228
98	Volcano	location	772
99	RadioHost	people	309
100	Castle	location	722

101	LacrossePlayer	people	335
102	TradeUnion	organization	1147
103	Species	biology	3006
104	Theatre	location	665
105	FashionDesigner	people	367
106	Hotel	location	901
107	Chef	people	395
108	Jockey	people	406
109	AmateurBoxer	people	408
110	WorldHeritageSite	location	662
111	DartsPlayer	people	436
112	SumoWrestler	people	433
113	TableTennisPlayer	people	426
114	Ambassador	people	446
115	BadmintonPlayer	people	461
116	Curler	people	460
117	MilitaryConflict	time	10357
118	Cardinal	people	399
119	Reptile	biology	5035
120	NationalCollegiateAthleticAssociationAthlete	people	523
121	Murderer	people	520
122	PublicTransitSystem	location	292
123	Lighthouse	location	1312
124	PokerPlayer	people	621
125	BroadcastNetwork	tv	377
126	Astronaut	people	619
127	SoapCharacter	fictional_universe	2133
128	SpeedwayRider	people	634
129	Amphibian	biology	6550
130	RecordLabel	business	1871
131	Park	location	1164
132	Economist	people	692
133	Religious	people	702
134	Engineer	people	729
135	VideoGame	cvg	14814
136	HollywoodCartoon	film	1439
137	HandballPlayer	people	765
138	GovernmentAgency	organization	1447
139	Fungus	biology	7640

140	NascarDriver	people	821
141	Non-ProfitOrganisation	organization	1902
142	FormulaOneRacer	people	824
143	ShoppingMall	location	2048
144	VolleyballPlayer	people	864
145	Airline	business	2909
146	Hospital	location	1691
147	RailwayLine	location	746
148	Senator	people	930
149	PoliticalParty	organization	3055
150	FictionalCharacter	fictional_universe	3310
151	PowerStation	location	1377
152	ReligiousBuilding	location	2220
153	Comedian	people	983
154	ChessPlayer	people	1051
155	Gymnast	people	1070
156	Country	location	1701
157	ComicsCharacter	fictional_universe	4392
158	Bird	biology	12227
159	Bridge	location	2861
160	Skier	people	1235
161	MilitaryStructure	location	1987
162	Architect	people	1276
163	Philosopher	people	1293
164	RacingDriver	people	1339
165	Model	people	1329
166	Island	location	3028
167	Judge	people	1554
168	AdultActor	people	1567
169	RailwayStation	location	4093
170	Organisation	organization	5122
171	BeautyQueen	people	1839
172	HistoricBuilding	location	3855
173	Anime	tv	770
174	President	people	2160
175	SoccerClub	sports	14006
176	ComicsCreator	people	2401
177	Monarch	people	2279
178	Swimmer	people	2453

179	WrittenWork	book	1023
180	FigureSkater	people	2507
181	Mollusca	biology	24214
182	GolfPlayer	people	2601
183	Wrestler	people	2604
184	Boxer	people	2666
185	MartialArtist	people	2675
186	Saint	people	2666
187	Congressman	people	2805
188	Play	book	900
189	ProtectedArea	location	6369
190	Stadium	location	6907
191	GaelicGamesPlayer	people	3006
192	MilitaryUnit	organization	10187
193	TennisPlayer	people	3343
194	Noble	people	3626
195	Lake	location	9242
196	Cyclist	people	3800
197	HistoricPlace	location	5564
198	Plant	biology	37258
199	EducationalInstitution	organization	11855
200	EthnicGroup	people	2863
201	TelevisionEpisode	tv	7056
202	ChristianBishop	people	4639
203	Airport	location	10313
204	Mountain	location	9905
205	CollegeCoach	people	5589
206	AustralianRulesFootballPlayer	people	5802
207	BasketballPlayer	people	6438
208	MusicGenre	music	598
209	School	organization	22375
210	Road	location	13875
211	GridironFootballPlayer	people	7004
212	BritishRoyalty	people	7034
213	Station	location	15091
214	City	location	19619
215	Magazine	book	2634
216	PopulatedPlace	location	18339
217	River	location	21498

218	Animal	biology	72508
219	Newspaper	book	4062
220	AcademicJournal	book	3440
221	RugbyPlayer	people	11024
222	IceHockeyPlayer	people	11478
223	Cricketer	people	11502
224	AmericanFootballPlayer	people	11835
225	Company	business	31860
226	SoccerManager	people	13321
227	TelevisionShow	tv	23423
228	Scientist	people	14534
229	Building	location	25291
230	Athlete	people	16963
231	Politician	people	18679
232	BaseballPlayer	people	19520
233	MilitaryPerson	people	23336
234	Artist	people	24682
235	Song	music	2809
236	MusicalArtist	people	37331
237	OfficeHolder	people	37832
238	Film	film	73335
239	Village	location	67191
240	Book	book	25114
241	SoccerPlayer	people	88568
242	Settlement	location	221982
243	Band	music	27535
244	Single	music	29441
245	Person	people	597566
246	Album	music	104958